王英瑛 乔小燕 吕廷华 编著

JSP Web
开发案例教程

清华大学出版社

JSP Web 开发案例教程

王英瑛　乔小燕　吕廷华　编著

清华大学出版社
北　京

内 容 简 介

本书全面、翔实地介绍应用 JSP 进行 Web 程序开发所需的各种知识和技能，主要内容包括：JSP 技术概述；HTML 标记语言和 JavaScript 脚本语言；JSP 语法基础；JSP 内置对象；JSP 与 JavaBean；JSP 访问数据库；在 JSP 中应用 Servlet 技术；JSP 的分页技术；文件上传下载技术；分层实现业务处理；JSTL 及 EL 等。通过一个实际的项目，以案例的方式介绍 JSP 程序设计技术，适合项目驱动、案例教学、理论与实践结合的教学方法，将知识讲解和技能训练有机结合，融“教、学、练”于一体。

本书可以作为普通高校计算机及相关专业“Web 程序设计”、“JSP 程序设计”、“动态网站制作”等课程的教材，同时也适合 JSP 的初学者和网站开发人员参考。阅读本书前读者最好熟悉 Java 编程语言、计算机网络等相关的基础知识。

图书在版编目（CIP）数据

JSP Web 开发案例教程/王英瑛，乔小燕，吕廷华编著. --北京：清华大学出版社，2013

ISBN 978-7-302-33334-0

Ⅰ. ①J… Ⅱ. ①王… ②乔… ③吕… Ⅲ. ①JAVA 语言－网页制作工具－教材 Ⅳ. ①TP312 ②TP393.092

中国版本图书馆 CIP 数据核字(2013)第 173617 号

责任编辑：刘　颖
封面设计：
责任校对：王淑云
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　张**：14　　**字　　数**：337 千字
版　　次：2013 年 9 月第 1 版　　**印　　次**：2013 年 9 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：

前 言

JSP(Java Server Page)是由Sun公司倡导，多家公司一起参与制定的一种动态网页技术标准。近年来JSP发展迅速，成为最引人注目的Web开发技术之一。JSP继承了Java的面向对象、跨平台等特性，特别是结合Servlet与JavaBean技术，使得页面代码与后台业务逻辑代码分离，解决了过去Web开发技术存在的不足，提高了工作效率。



本书特点及知识结构

本书为项目驱动，主要特色是以贯穿项目"通知公告发布系统"为主线，通过实际开发Web项目系统地介绍JSP的知识。主要分为三部分，辅助知识：JSP开发环境、开发工具、HTML、CSS、JavaScript和JSP的基本常识；核心技术：JSP语法、内置对象、JDBC、JavaBean和Servlet；扩展技术：MVC、上传下载、EL、JSTL与框架技术的简介。



本书面向的读者

本书可以作为普通高校计算机及相关专业"Web程序设计"、"JSP程序设计"、"动态网站制作"等课程的教材，同时也适合JSP的初学者和网站开发人员参考。阅读本书前读者最好熟悉Java编程语言、计算机网络等相关的基础知识。



本书结构

本书每章用贯穿项目及一些辅助的示例讲解知识点；其后针对该知识点设计相应的上机练习，上机练习中让读者明确需求并辅以实现思路和关键代码；接下来是总结，简要列举本章重要知识点；最后是作业，通过选择题、简答题和编程题让读者对本章知识加以熟练掌握。全书共分10章，各章具体内容如下：

第1章　Hello JSP，　　　　简要介绍JSP，以及如何搭建开发JSP运行环境。
第2章　静态网页开发基础，　介绍HTML、CSS和JavaScript语言。
第3章　JSP基础，　　　　　介绍JSP的工作原理和页面元素。
第4章　JSP数据库应用开发，介绍数据库的基础知识、SQL语言和JDBC。
第5章　JSP中的JavaBean，　介绍JavaBean的相关知识。

第 6 章　JSP 内置对象，　　介绍常用的 JSP 内置对象。
第 7 章　Servlet 技术，　　介绍 Servlet 相关知识。
第 8 章　MVC 设计模式，　　介绍开发基于 MVC 设计模式的应用程序。
第 9 章　JSP 开发业务应用，　介绍 JSP 分页技术和文件上传下载。
第 10 章　JSP 高级程序设计，　简要介绍 El、JSTL 和框架技术。

技术支持

本书“通知公告发布系统”及示例的相关代码都已通过测试，同时制作了多媒体课件。如需要源码及课件，可以从“www.tup.tsinghua.edu.cn”免费下载。

最后感谢您选择并阅读本书。由于作者水平有限，书中难免有错误、疏漏之处，敬请读者批评指正。

编　者
2013 年 5 月

目　录

第 1 章　Hello JSP

本章学习目标

- 掌握静态网页和动态网页
- 掌握 B/S 架构
- 掌握使用 URL 访问动态网页
- 熟练搭建 JSP 开发运行环境
- 掌握开发 JSP 动态网站的基本步骤

JSP(Java Server Pages)是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准,是基于 Java 语言的一种 Web 应用开发技术。相对其他 Web 应用开发技术,它具有脱离硬件平台束缚、编译后运行等优点,已成为 Internet 上的主流 Web 应用开发技术之一。

1.1　动 态 网 页

1.1.1　动态网页 PK 静态网页

静态网页是指纯粹 HTML 代码格式的网页,由静态网页组成的不具备交互性的网站称为静态网站。当我们上网时,会看到某些静态商业网站,只是展示、罗列内容。而动态网站却能“动”起来,例如实现添加、修改、删除信息、在线搜索等功能,实现与用户真正的互动。

由于静态网页内容是固定的,制作完成后,就不能随意更改。当我们想要修改静态网页时,必须通过专用的网页制作工具,比如 Dreamweaver、FrontPage 等,而且只要修改了网页中的一个字符或者一个图片,就必须重新将服务器上的网页进行覆盖。随着 Web 应用的发展,静态页面不能迅速地进行添加和修改等操作,也不能提供个性化和定制化的服务等不足逐渐显现,与此相对应地,动态网页技术逐渐发展并得到广泛的应用。

那么什么是动态网页呢?

1.1.2　什么是动态网页

动态网页是指其页面信息可以根据需求或者用户的浏览状况,实现与用户交流和页面信息自动更新的网页。例如,当我们浏览论坛网站时,只能看到帖子的浏览页面;想要发表

自己的帖子，必须先登录论坛，变换浏览状况，发表帖子后，呈现“删除”、“修改”等多种操作提示供我们选择，可以对所发表的帖子进行处理。另外，我们在不同的时间登录论坛，看到的帖子列表也是不同的。

在日常生活中，我们经常会用到很多动态网页，例如百度、淘宝网等。当我们在百度的搜索栏中输入关键字“JSP”时，页面就会自动排列出所有有关“JSP”的网址链接，如图 1-1 所示。

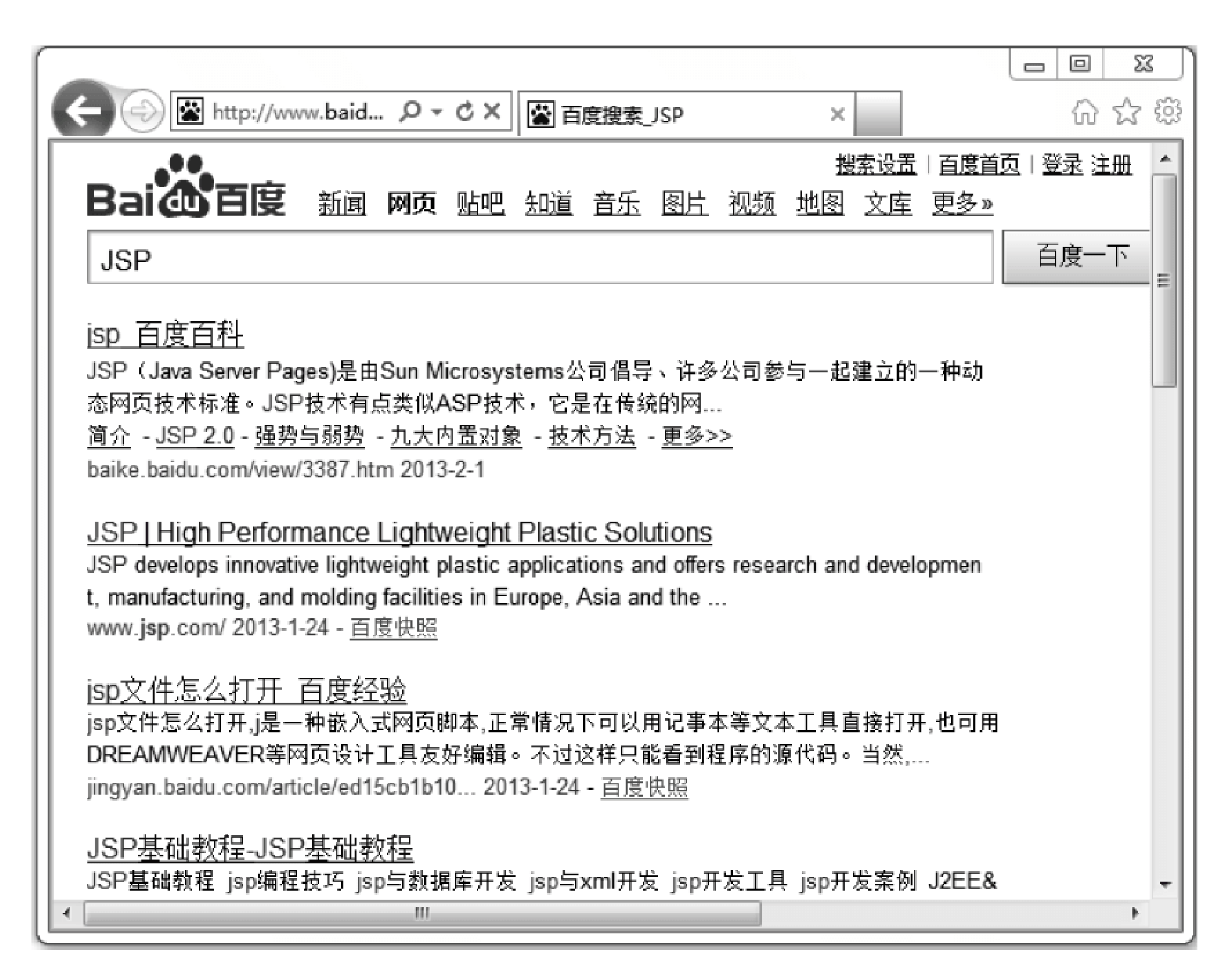


图 1-1 搜索“JSP”的百度网页

而当我们在淘宝网的搜索栏中输入关键字“移动硬盘”时，页面就会自动排列出所有包含“移动硬盘”的网址链接，如图 1-2 所示。

通过图 1-1 和图 1-2 的效果展示，我们来总结一下动态网页的特点：

- **交互性**：网页会根据用户的要求和选择而动态改变和显示内容。
- **自动更新**：无需手动操作，动态网页以数据库技术为基础，根据需求自动生成新的页面，可以大大节省工作量。
- **随机性**：即当不同的时间、不同的人访问同一网址时会产生不同的页面效果。



问题：动态网页是静态网页的替代品吗？

答：静态网页和动态网页各有特点，网站采用动态网页还是静态网页主要取决于网站的功能需求和网站内容的多少。如果网站功能比较简单，内容更新量不是很大，采用静态网页的方式会更简单，反之，则采用动态网页技术来实现。静态网页是网站建设的基础，静态网页和动态网页之间也并不矛盾，为了使网站能适应搜索引擎检索的需要，即使采用动态网站技术，也可以将网页内容转化为静态网页发布。

动态网站也可以采用静动结合的原则，适合采用动态网页的地方用动态网页，如果有必要使用静态网页，则可以考虑用静态网页的方法来实现，在同一个网站上，动态网页内容和静态网页内容同时存在也是很常见的事情。



图 1-2 搜索“移动硬盘”的淘宝网页

1.1.3 动态网页的实现

通过如图 1-1 所示的界面可以看到，在百度的数据库中保存了成千上万条符合“JSP”的查询结果。在页面上也分页排列出了所有包含“JSP”的网址链接。

那么究竟怎样才能开发出像百度、淘宝网这样的动态网页呢？

动态网页需要使用服务器端脚本语言，例如 JSP 等。本书就是以 JSP 技术为核心，结合使用 JSP 开发动态网站过程中应用的其他相关技术，向大家介绍如何开发 Web 应用系统。这需要一个学习的过程，本书将会采用案例贯穿、循序渐进的方式引领大家一步一步掌握动态网页的开发技术。

1.2 B/S 架 构

1.2.1 B/S 架构技术

什么是 B/S 架构技术呢？

B/S(Browser/Server，浏览器/服务器)是互联网普及与大规模应用后的一种网络结构模式。这种模式统一了客户端，将系统功能实现的核心部分集中到服务器上，简化了系统的开发、维护和使用。基于 B/S 架构的 Web 应用程序由于不再受到安装客户端的限制，访问极其简便，因此越来越多地被企业采用。例如在全球任何一个安装浏览器的计算机上，我们都可以访问百度网站，如图 1-3 所示，这就是基于了 B/S 架构。

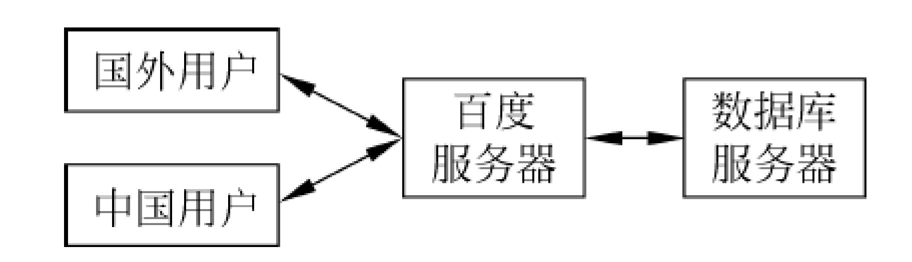


图 1-3　基于 B/S 架构示例

在 B/S 架构下，应用系统完全放在应用服务器上，并通过应用服务器同数据库服务器进行通信。在客户机上无需安装任何客户端软件，系统界面是通过浏览器来展现的。对于用户而言，只需要能连接 Internet，安装完浏览器就可以访问系统了，如图 1-4 所示。对于程序开发，维护人员来说，无论用户身处何地，所要做的就是对服务器的代码进行更新和维护。

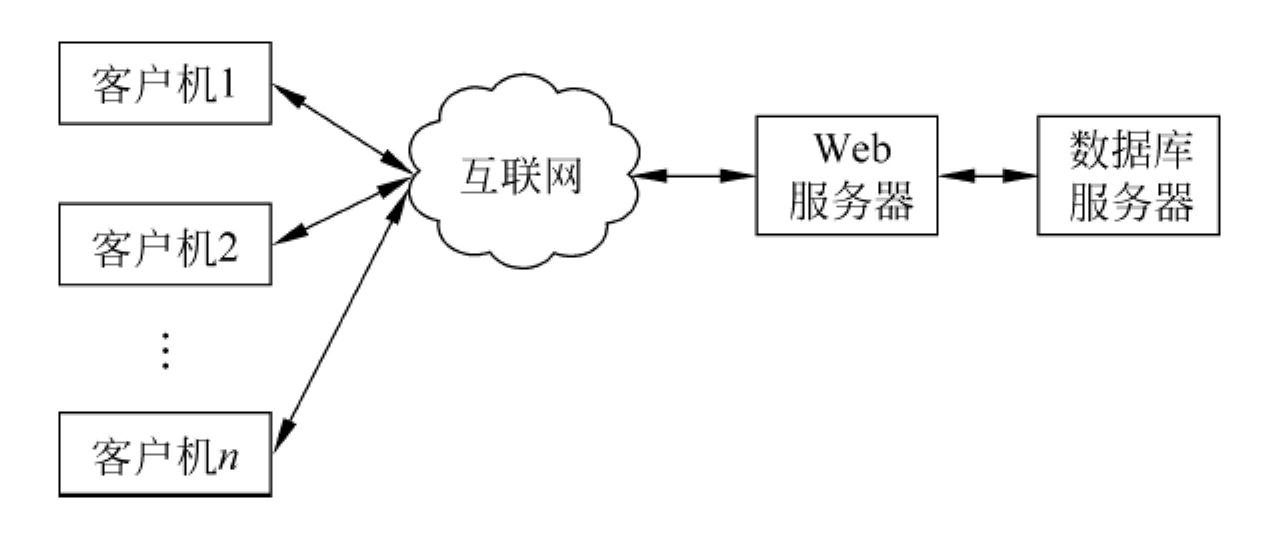


图 1-4　B/S 架构图

B/S 架构的优点是：

- 分布性强，可以随时随地进行查询、浏览等业务处理。
- 业务扩展方便，通过增加页面即可增加服务器功能。
- 维护简单，只需要改变网页，即可实现所有用户的同步更新。
- 共享性强。

缺点是：

- 个性化特点明显降低，无法实现具有个性化的功能要求。
- 操作是以鼠标为最基本的操作方式，无法满足快速操作的要求。
- 页面动态刷新，响应速度明显降低。
- 功能弱化，难以实现传统模式下的特殊功能要求。

1.2.2 B/S 架构的工作原理

了解了 B/S 架构及其优缺点,我们再来深入了解一下 B/S 技术的相关特点,看看基于 B/S 架构的应用程序是如何工作的。在 B/S 架构中,浏览器端与服务器端采用请求/响应模式进行交互,工作原理如图 1-5 所示。

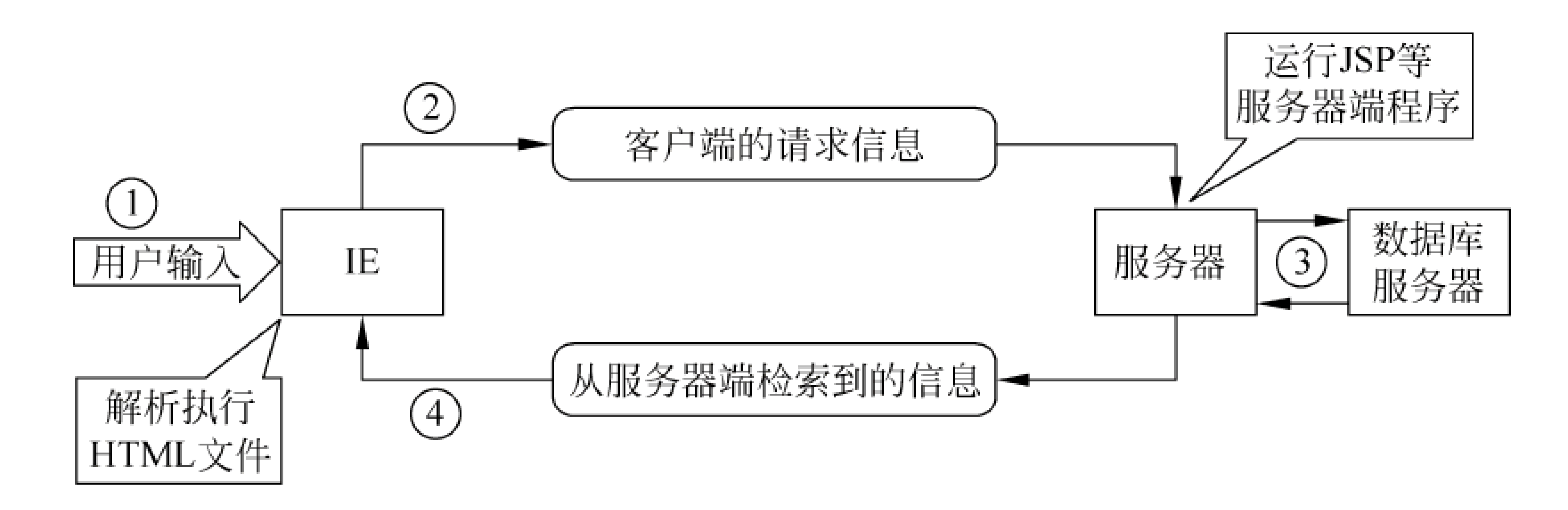


图 1-5　请求/响应模式

图 1-5 展示了 B/S 架构的工作流程,下面我们就逐步分解其中的每个环节。

1. 浏览器接受用户的输入,例如,一个用户在浏览器窗口中输入用户名、密码,单击“登录”按钮(全书中如果没有说明所操作的是鼠标的左键还是右键,就默认所操作的是左键)。

2. 浏览器向服务器端发送请求:浏览器把请求消息(包含用户名、密码等信息)发送到服务器端,等待服务器端的响应。

3. 数据处理:服务器端通常使用服务器端脚本语言如 JSP 等来访问数据库,查数据,并获得查询结果。

4. 发送响应:服务器端向浏览器发送响应消息(一般是动态生成的 HTML 页面),并由浏览器解释 HTML 文件,呈现结果界面。

我们已经了解了 B/S 技术及开发 B/S 架构应用系统的工作流程,那么应用系统开发完成后,该如何访问呢?接下来我们将学习如何使用 URL 访问动态网页。

1.3 使用 URL 访问动态页面

1.3.1 什么是 URL

URL(Uniform Resource Locator)其中文意义为统一资源定位符,它是互联网上标准的资源地址,是用于完整地描述 Internet 上网页和其他资源地址的一种标识方法。简单地说,URL 就是我们常说的“网址”。



URL 是为了能够使客户端程序查询不同的信息资源时有统一的访问方法而定义的一种地址标识方法。在 Internet 上所有资源都有一个独一无二的 URL 地址。我们可以通过在浏览器地址栏输入 URL 实现对网页的访问。

1.3.2 URL 的组成

URL 地址是来告诉浏览器要访问的服务器地址，那么浏览器是怎么解读这个地址的呢？首先我们来看一下 URL 地址的组成。

在本书中会经常使用的一个 URL：

http://localhost:8080/Notice/page/background/index.jsp，通过这个 URL 地址，可以分析得出它的组成结构。

- HTTP 协议：HTTP（HyperText Transmission Protocol）即超级文本传输协议，该协议支持简单的请求和响应会话，当用户发送一个 HTTP 请求时，服务器就会用一个 HTTP 响应做出回答。用一句简单的话来描述就是：对于 Web 服务器就必须要使用 HTTP 协议。
- 服务器域名或 IP：在前面讲解 B/S 架构时，我们知道了 Web 应用的运行是基于 Web 服务器的，由服务器收集用户的请求，经过处理再返回响应到浏览器端。这也就是说我们需要访问 Web 服务器的服务，那么 localhost 就代表着服务器的地址，当然在开发阶段通常都是以开发机作为服务器，因此服务器的地址就可以使用 localhost 来进行标识，也可以使用 127.0.0.1 或者用实际的 IP 地址来替代。
- 端口号：端口是服务器用于内外部通信的通道，当从外部访问服务器时就需要通过指定的通道来访问。不同的服务器有着各自不同的默认开发端口，开发人员可以根据实际需要进行修改。
- 路径：路径实际上包含两层含义。以 Notice/page/background/index.jsp 为例，Notice 代表的是 Web 应用对外发布时对应的上下文路径，即 Web 应用的根目录 WebRoot 文件夹，而/page/background/index.jsp 代表网页存放的具体位置。

通过上述分析，可以总结得出最常用的 URL 的组成部分如下。

第一部分：协议，指使用的传输协议，如最常用的 HTTP 协议（目前 WWW 中应用最广的协议）。

第二部分：主机 IP 地址（有时包含端口号），指请求的服务器的 IP 地址，这个地址是唯一的。当然，也可以使用服务器名称来代替 IP 地址发送请求。

第三部分：路径（包含请求的资源），指由零或多个“/”符号隔开的字符串，一般用来表示主机上的一个目录或文件地址等。而请求的资源指请求内容的名字，可以是一个 HTML 页面，也可以是一个 Servlet、图像等服务器能提供的资源。

提 示

第一部分和第二部分之间用“://”符号隔开，第二部分和第三部分用“/”符号隔开。其中，第一部分和第二部分是不可缺少的，第三部分有时可以省略。当第三部分省略时，大多数服务器会默认访问系统的欢迎页面。

现在我们已经掌握了 B/S 架构应用系统的工作原理及流程，也知道了如何正确编写 URL，那么接下来是不是就可以开发 Web 应用系统了呢？工欲善其事必先利其器，我们首先来搭建 JSP 的开发环境。

1.4 搭建 JSP 运行环境

使用方便、高效快捷的 JSP 开发环境，对于学习开发 JSP 的读者来说事半功倍。目前，较流行的 JSP 开发平台和工具主要包括 JDK、Tomcat、MyEclipse 等。其中，JDK（Java Developer Kit）是 Java 开发工具包；Tomcat 是 Web 服务器；MyEclipse 是一套十分优秀的开发工具，实际上就是 JSP 应用开发的可视化 IDE。编程人员首先利用 MyEclipse 平台，开发 Web 项目。而 Web 项目若能让别人访问，就必须将项目部署到服务器上才行，Tomcat 就是这样一个轻量级的服务器。我们开发的是包含 JSP 项目的 Web 项目，JSP 是在 HTML 文件中插入 Java 代码，而 JDK 则包含了 Java 的运行环境、工具及基本库，是整个 Java 的核心，也是运行 JSP 不可或缺的环境。下面就分别讲解 JDK、Tomcat 和 MyEclipse 的安装与配置。

1.4.1 JDK 的安装与配置

JDK 是 Sun 公司免费提供的 Java 语言工具，它包含了 Java 开发中所必需的开发工具和 Java 运行环境（JRE-Java Run Environment），是 Java 应用程序开发的基础。由于 JSP 本身执行的计算机语言就是 Java，因此，想要开发运行 JSP 程序也需要 JDK 的开发环境。JDK 目前的最新版本是 J2SE7.0(1.7)，安装文件可以通过

http://www.oracle.com/technetwork/java/javase/downloads/index.html

进行下载，下载的文件名为 jdk-7u9-windows-i586.exe，大小为 90470KB，如图 1-6 所示。

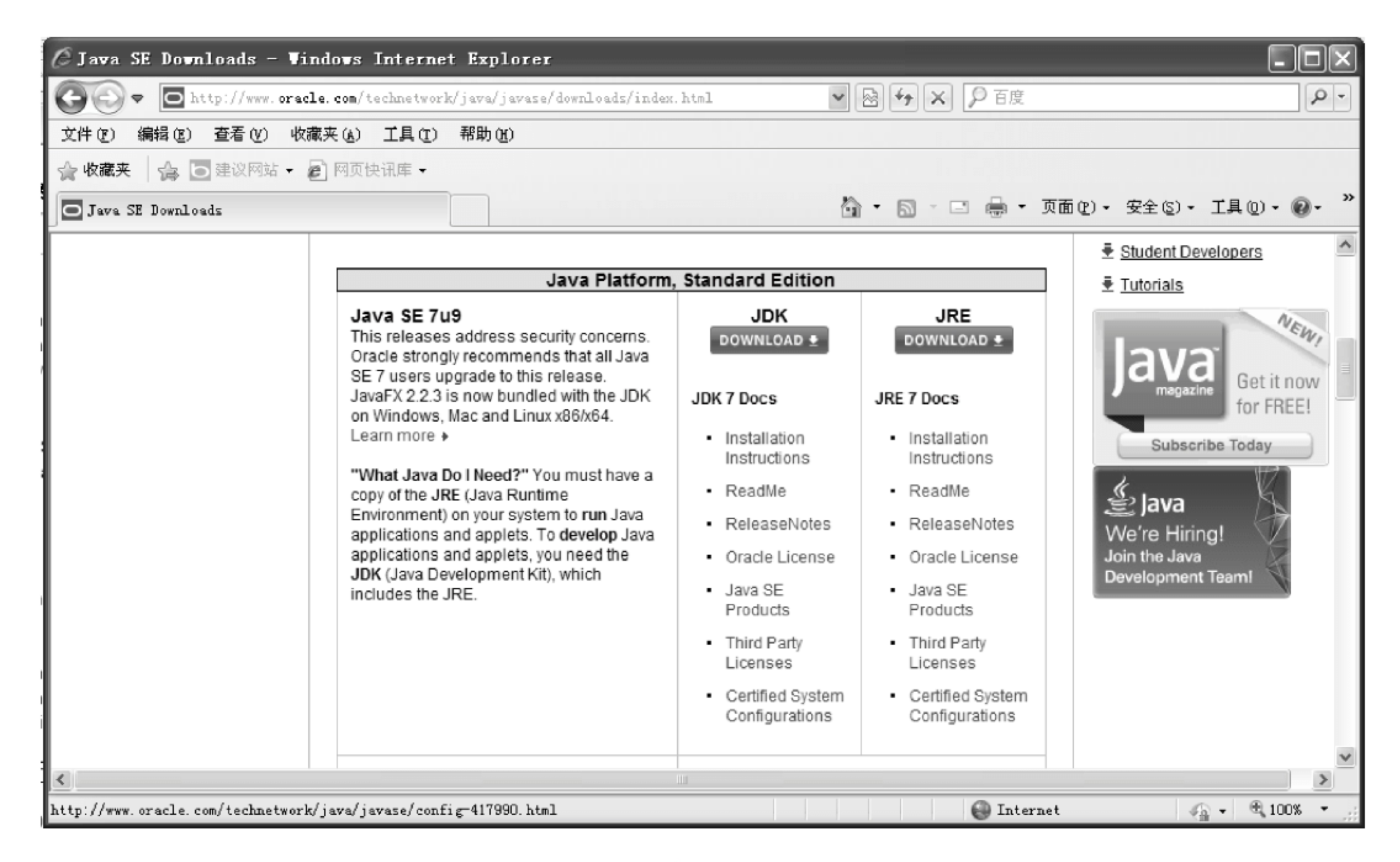


图 1-6 下载 J2SE1.7 网页

1. JDK 的安装

下载 JDK 后就可以安装 JDK 了，下面我们介绍安装步骤：

(1) 双击 JDK 的安装文件“jdk-7u9-windows-i586.exe”，弹出如图 1-7 所示的界面。



图 1-7　JDK 的安装界面

(2) 单击“下一步”按钮，弹出如图 1-8 所示的自定义安装界面，选择安装路径和安装的其他配置，在这里采用默认的设置。



图 1-8　JDK 自定义安装界面

(3) 单击“下一步”按钮，进入选择 JRE 安装路径的界面，如图 1-9 所示，设置 JRE 的安装路径，或根据需求更改路径。单击“下一步”按钮继续安装，至安装成功。如图 1-10 所示为 JDK 安装成功的页面。

2. JDK 的配置与测试

安装完 JDK 后，需要设置环境变量及测试 JDK 配置是否成功，具体步骤如下：



图 1-9 设置 JRE 的安装路径



图 1-10 JDK 安装成功

(1) 在“我的电脑”上单击鼠标右键，在弹出的快捷菜单中，选择“属性”选项(若是 Win7 操作系统，则在“计算机”上单击鼠标右键，在弹出的快捷菜单中选择“属性”选项，打开“高级系统设置”)。在打开的“系统属性”对话框中选择“高级”选项卡，如图 1-11 所示。

(2) 单击“环境变量”按钮，打开“环境变量”对话框。在这里可以添加针对单个用户的“用户变量”和针对所有用户的“系统变量”，如图 1-12 所示。



图 1-11 系统属性设置



图 1-12 环境变量设置

(3) 单击“系统变量”区域中的“新建”按钮，弹出“编辑系统变量”对话框。在对话框的“变量名”文本框中输入“JAVA_HOME”，在“变量值”文本框中输入 JDK 的安装路径 D:\Program Files\Java\jdk1.7.0_09，单击“确定”按钮，完成环境变量 JAVA_HOME 的配置，如图 1-13 所示。



图 1-13 新建系统变量设置

(4) 在系统变量中查看 PATH 变量，如果不存在，则新建变量 PATH，否则选中该变量，单击“编辑”按钮，打开“编辑系统变量”对话框，在该对话框的“变量值”文本框的起始位置添加“%JAVA_HOME%\bin;”。

(5) 单击“确定”按钮返回到“环境变量”对话框。在系统变量中查看 CLASSPATH 变量，如果不存在，则新建变量 CLASSPATH，变量值为%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar。

(6) JDK 程序的安装和配置完成后，可以测试 JDK 是否能够在计算机上运行。选择“开始”→“运行”命令，在打开的“运行”窗口中输入 cmd 命令，进入到 DOS 环境，在命令提示符后面直接输入“javac”，按下 Enter 键，系统会输出 javac 的帮助信息，如图 1-14 所示。这说明已经成功配置了 JDK，否则需要仔细检查上面步骤的配置是否正确。



图 1-14 测试 JDK 安装与配置是否成功

1.4.2 Tomcat 的安装、运行与目录结构

Tomcat 服务器是一个免费开源的 Web 应用服务器，我们可以直接从 Apache 的官方网站 http://tomcat.apache.org/进行下载，如图 1-15 所示。目前其最新版本是 7.0，下载的文件名为“apache-tomcat-7.0.33.exe”。Tomcat 是一个轻量级的应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。

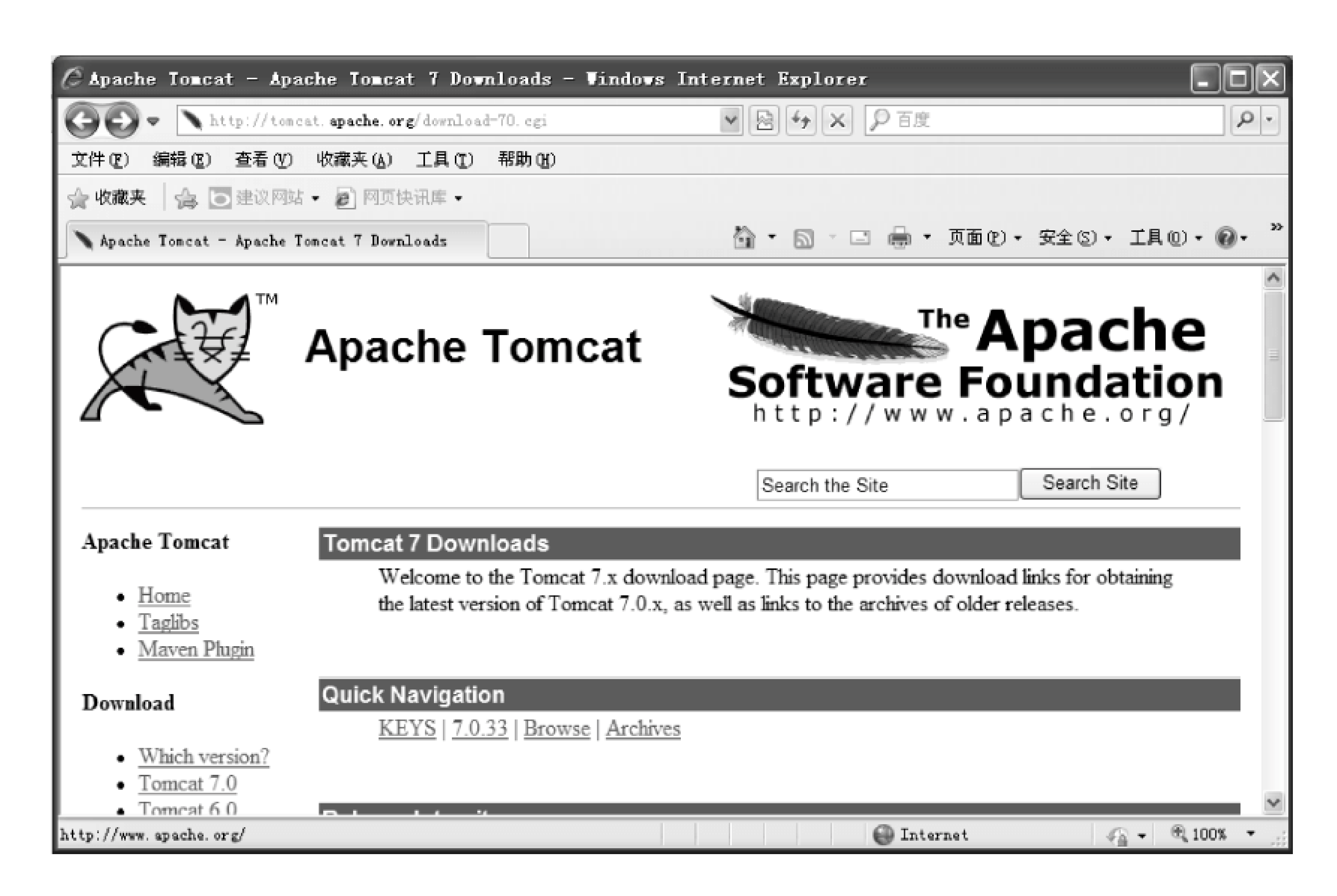


图 1-15 下载 Tomcat 7.0

1. Tomcat 的安装

安装 Tomcat 服务器，具体步骤如下。

（1）双击 apache-tomcat-7.0.33.exe 文件开始安装。在弹出的安装向导对话框中，单击“Next”按钮，将弹出如图 1-16 所示的许可协议对话框。

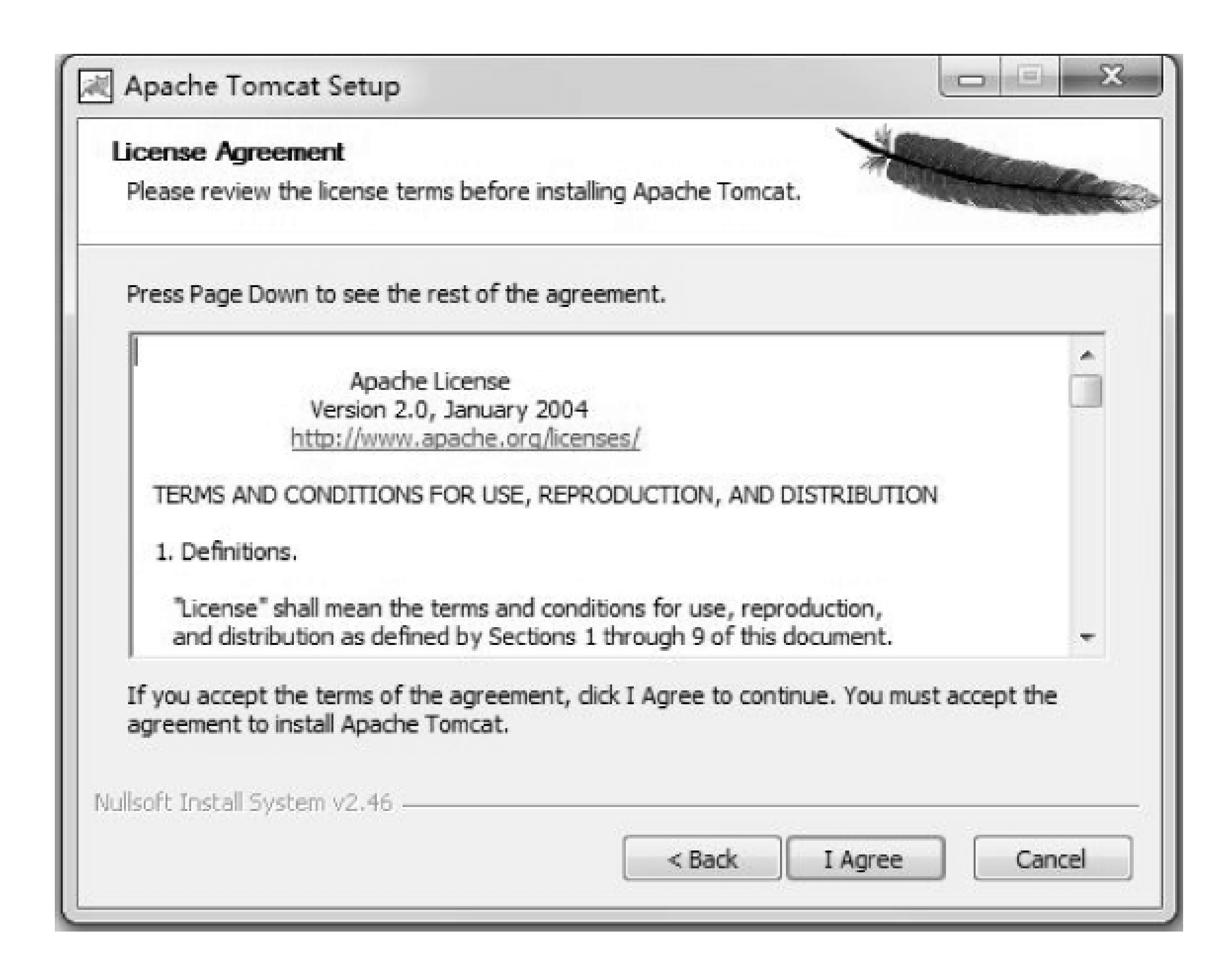


图 1-16 接受 Tomcat 使用协议

（2）单击“I Agree”按钮，接受许可协议，出现如图 1-17 所示的选择组件对话框，选择要安装的 Tomcat 组件，然后单击“Next”按钮。

（3）在图 1-18 所示的配置对话框中，Server Shutdown Port 指定 Tomcat 监听 shutdown 命令端口，默认为 8005；第一个 Connector Point 定义了一个 HTTP Connector，默认端口 8080 接收 HTTP 请求；第二个 Connector Point 定义了一个 JD Connector，默认 8009 端口接收由其他服务器转发过来的请求；接下来需要设置服务器的 Tomcat 后台登录名和密码。这里采用默认的安装。

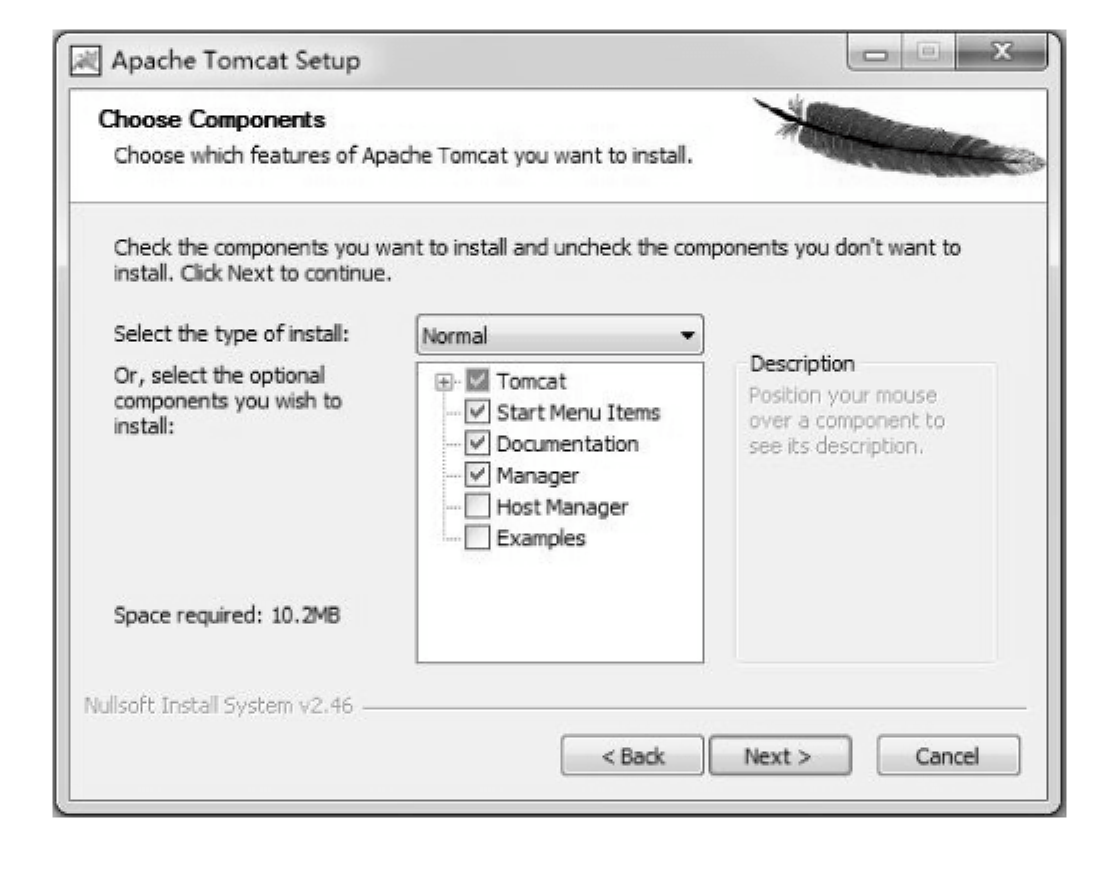


图 1-17 设置 Tomcat 的安装类型

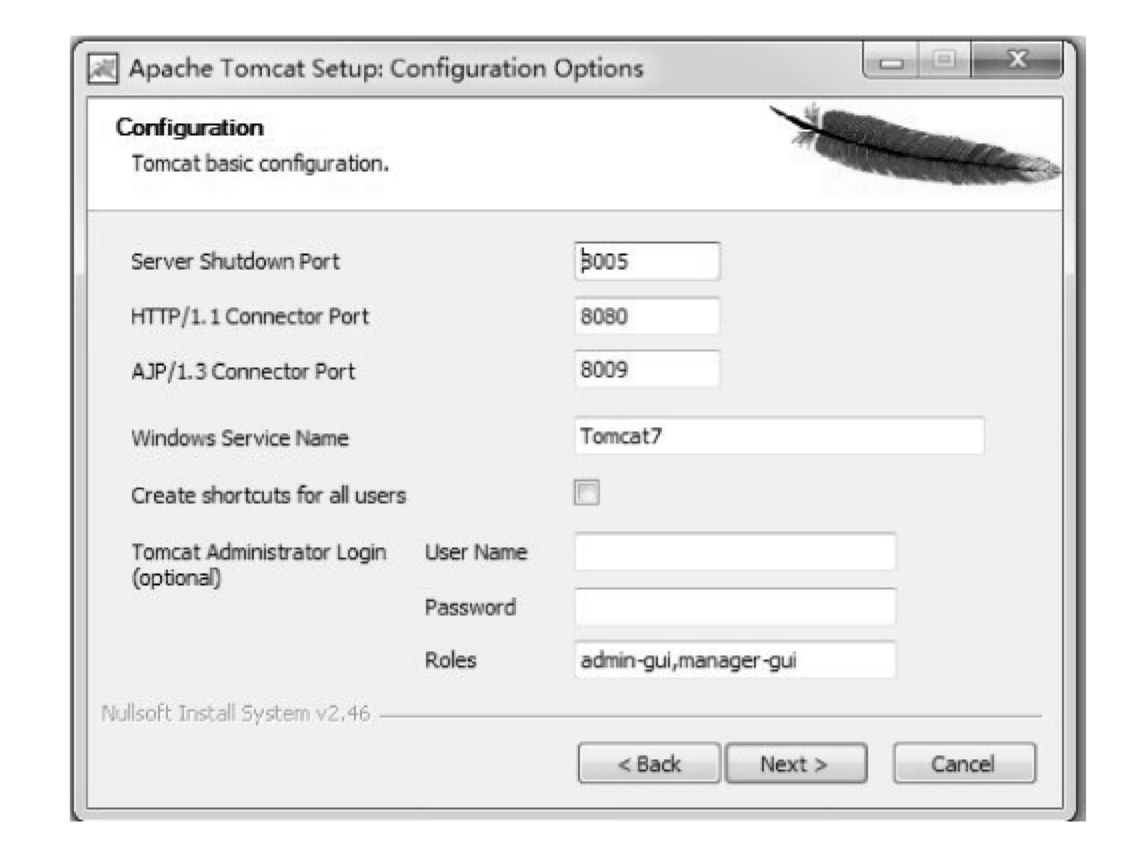


图 1-18 设置 Tomcat 端口号等

（4）单击"Next"按钮，将弹出如图 1-19 所示的对话框，一般情况下安装程序可以自动找到 Java 虚拟机路径设置。

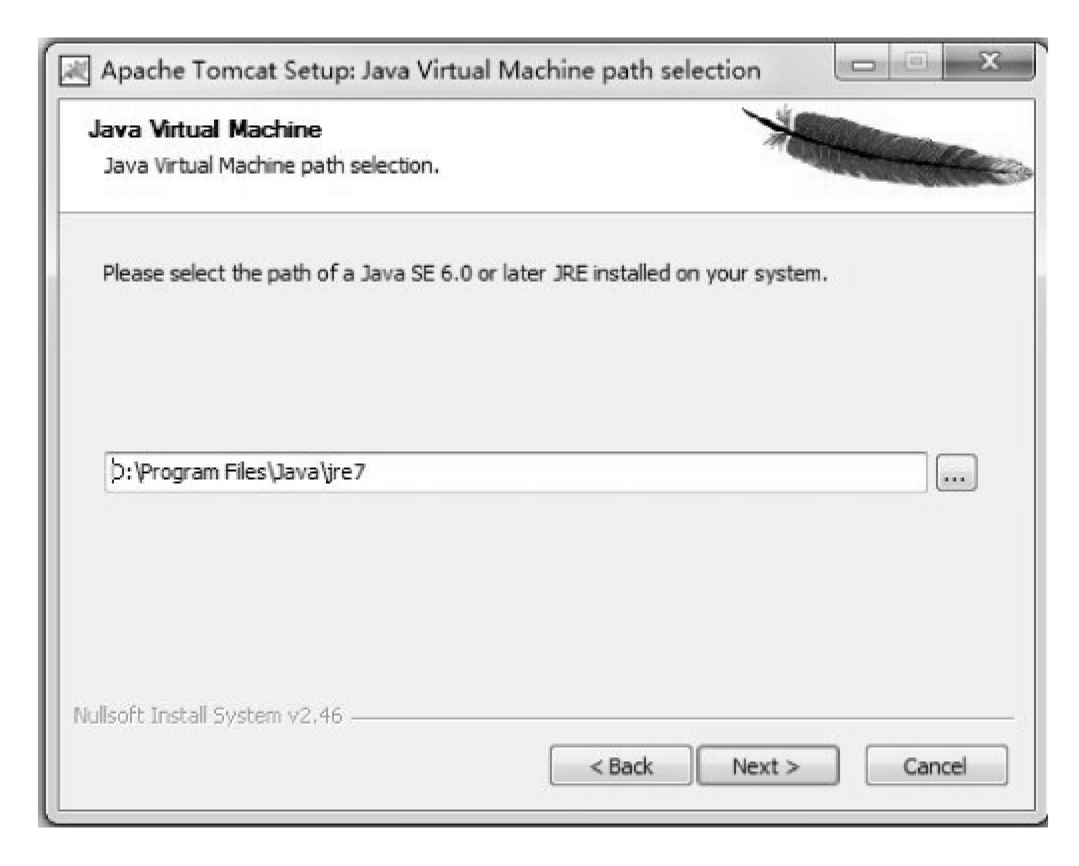


图 1-19 Java 虚拟机路径设置

（5）然后单击"Next"按钮，弹出设置 Tomcat 安装路径对话框（图 1-20），单击"Install"按钮开始安装。在弹出的安装对话框中单击"Finish"按钮，完成安装，如图 1-21 所示。

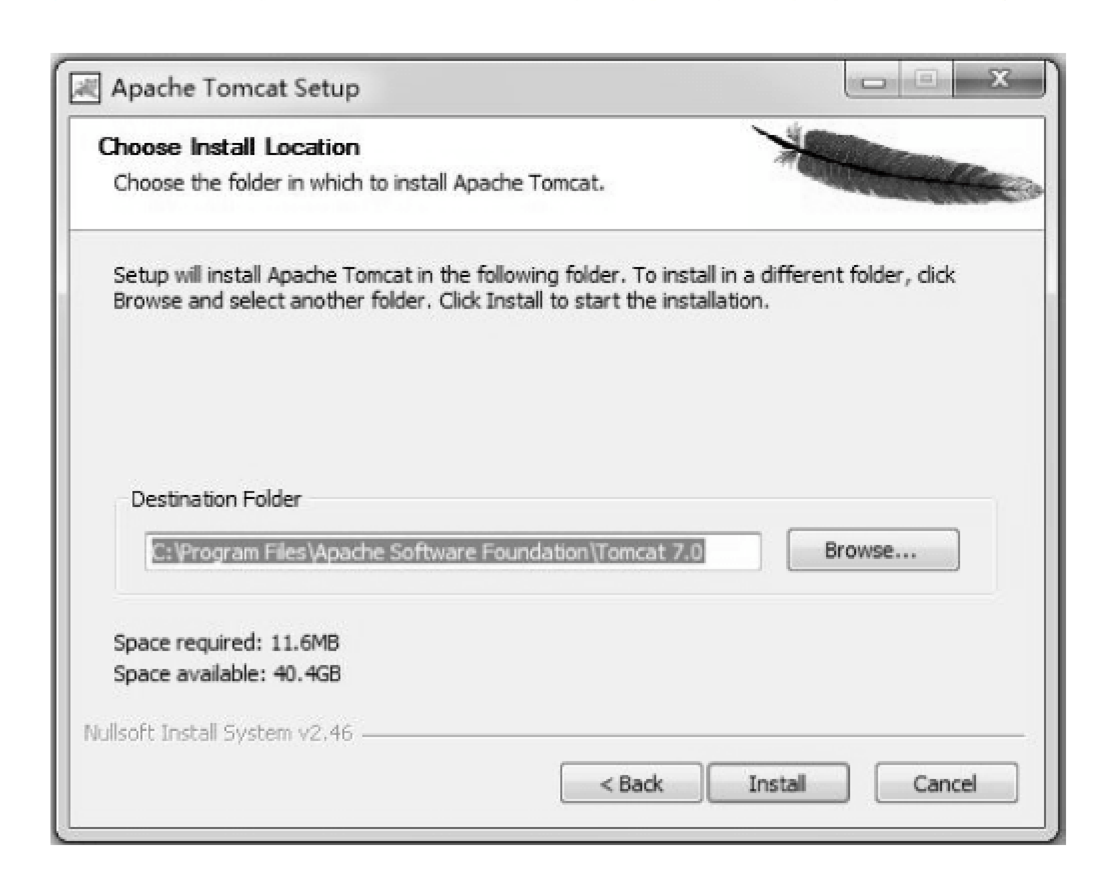


图 1-20 设置 Tomcat 安装路径



图 1-21 安装 Tomcat

2. Tomcat 的运行

Tomcat 安装完成后，就可以运行该服务器，具体步骤如下。

（1）在开始菜单中选择“开始”→“程序”→Apache Tomcat 7.0--Configure Tomcat 选项，弹出启动 Tomcat 服务器的界面，该界面可以对 Tomcat 的一些参数进行配置，一般采用默认方式。如图 1-22 所示。

（2）单击“Start”按钮，启动 Tomcat 服务器，如图 1-23 所示。

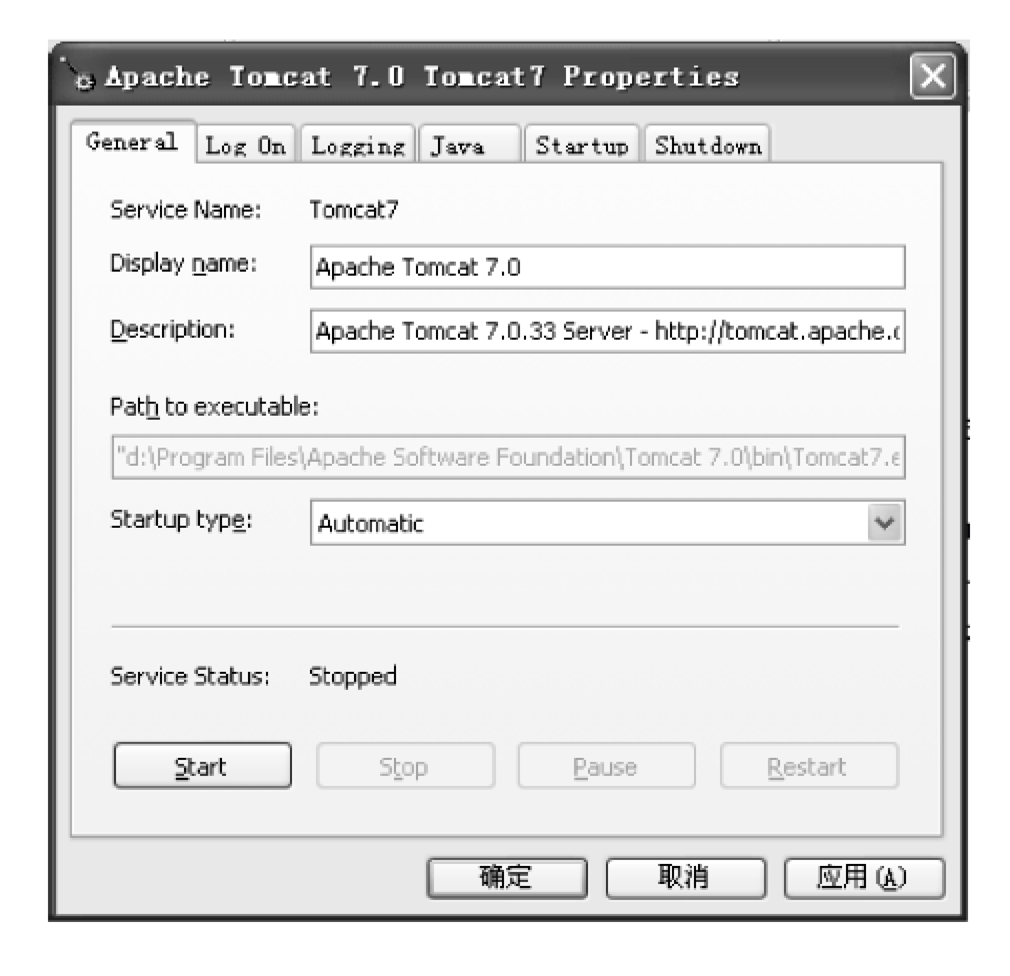


图 1-22 配置 Tomcat 参数

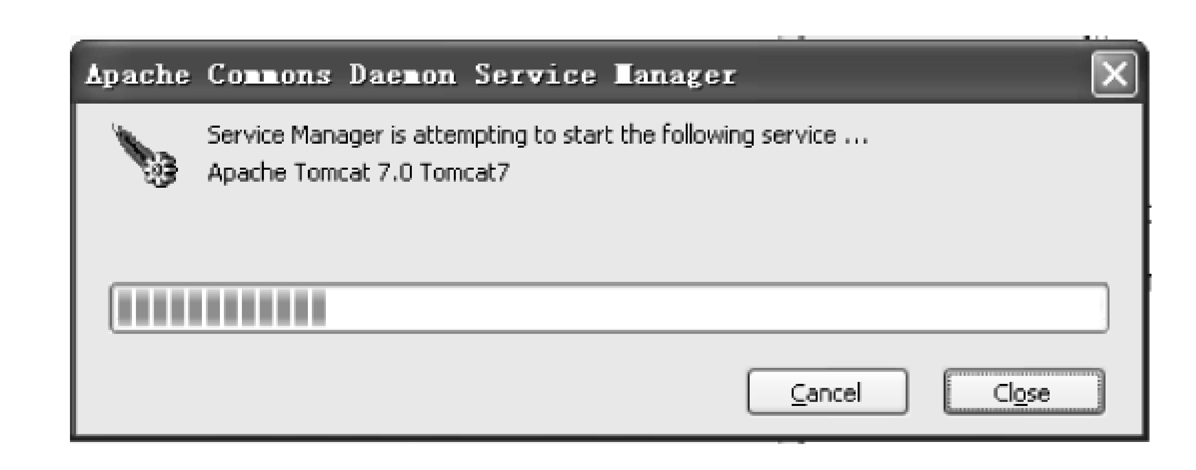


图 1-23 Tomcat 服务器启动

（3）打开 IE 浏览器，在地址栏中输入“http://localhost:8080”，运行结果如图 1-24 所示。说明 Tamcat 安装成功。

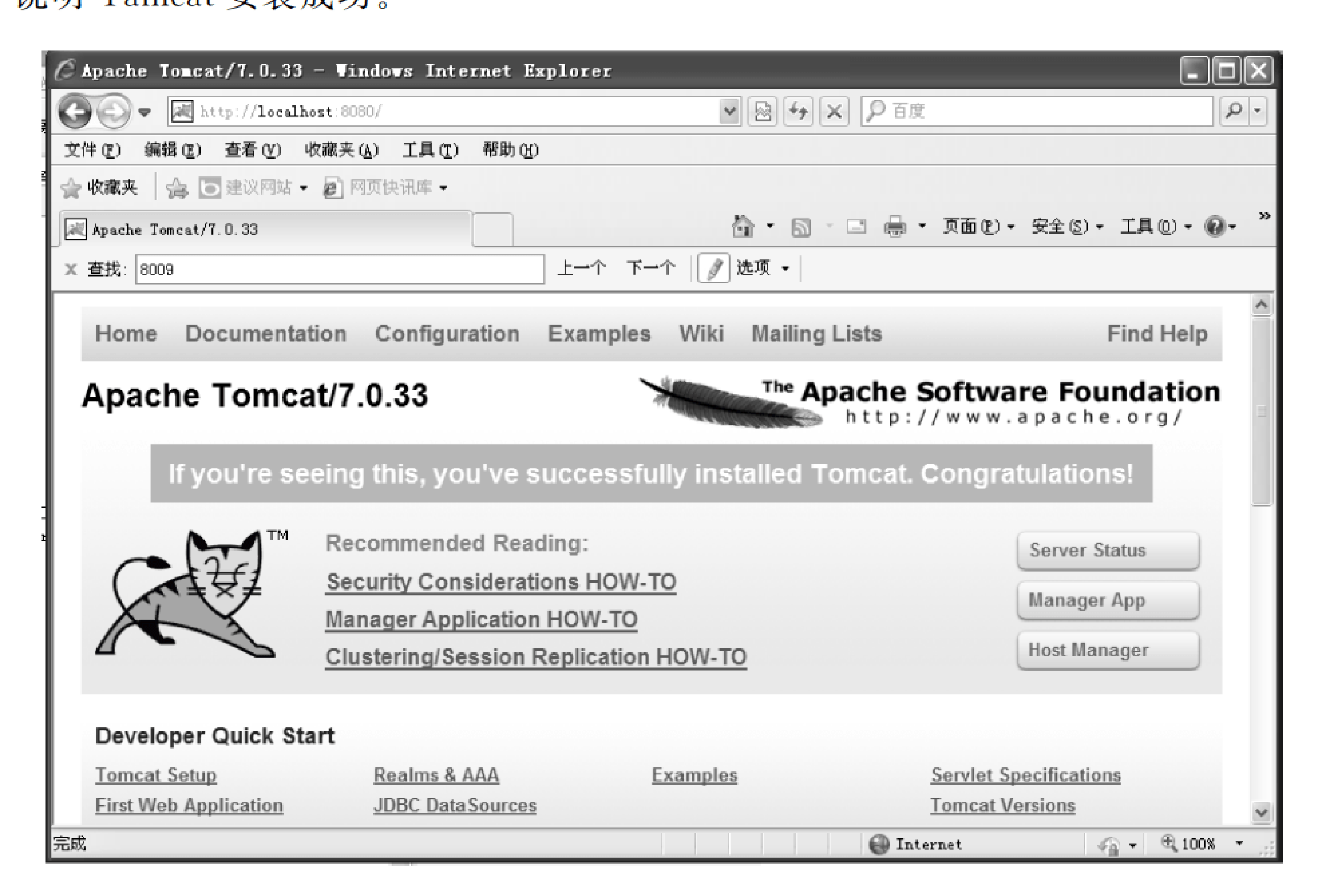


图 1-24 Tomcat 启动成功页面

提 示

若没有出现如图 1-24 所示的欢迎页面,可以检查 Tomcat 安装目录下的 webapps 文件夹下是否有 ROOT 文件夹,及该文件夹下是否不为空,如果为空,则安装过程中如图 1-17 时没有选择 HostManager,请重新安装并选择该项。

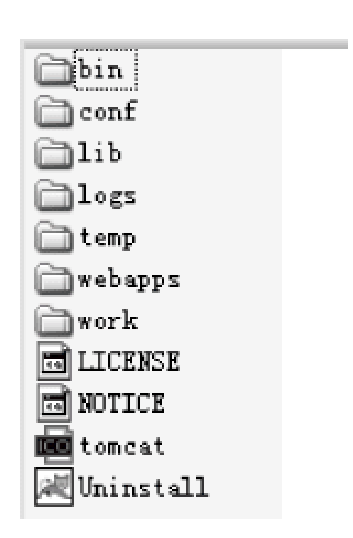


图 1-25 Tomcat 目录结构

3. Tomcat 目录结构

Tomcat 服务器安装完毕后,打开 Tomcat 的安装路径,会看到如图 1-25 所示的目录结构。

不同的目录具有不同的作用,需要注意的是 Tomcat 在不同版本之间的目录结构略有区别,本书以 Tomcat 7.0 的目录结构为例来介绍,每个目录的功能描述见表 1-1。

表 1-1 Tomcat 目录结构表

目 录	说 明
/bin	存放各种平台下用于启动和停止 Tomcat 的脚本文件
/conf	存放 Tomcat 服务器的各种配置文件,其中最重要的是 server. xml
/lib	存放 Tomcat 服务器所需的各种 JAR 文件
/logs	存放 Tomcat 的日志文件
/temp	Tomcat 运行时用于存放临时文件
/webapps	Web 应用的发布目录
/work	Tomcat 把由 JSP 生成的 Servlet 放在此目录下

/conf 下的 server. xml 文件是 Tomcat 服务器重要的配置文件,在该文件中有诸多元素的配置可影响到服务器的性能。例如,通过 Tomcat 开放的端口号 8080,可以访问应用程序,如果该端口号一旦被占用,将无法再继续访问,此时只需对 server. xml 进行更改,就可以实现端口号的更改。

示例1-1

```
/**
 * 修改前配置
 */
<Connector port = "8080" protocol = "HTTP/1.1"
                connectionTimeout = "20000"
                redirectPort = "8443" />
```

```
/**
 * 修改后配置
 */
<Connector port = "8081" protocol = "HTTP/1.1"
                connectionTimeout = "20000"
                redirectPort = "8443" />
```

Tomcat 服务重启后，在浏览器中输入 http://localhost:8081/，出现如图 1-25 所示的欢迎页面，则测试修改成功。

1.4.3 MyEclipse 的使用

MyEclipse 企业级工作平台(My Eclipse Enterprise Workbench)是由 Genuitec 公司推出的商业版 J2EE 开发平台，提供一个月的免费试用期，它的官方网站是 http://www.myeclipseide.com，可在其主页中(见图 1-26)下载安装包。请根据需求，下载安装，此处不再赘述 MyEclipse 的安装过程。



图 1-26　MyEclipse 官网


说　明

MyEclipse 和 Eclipse 的比较

我们知道，Eclipse 是一个免费的 Java 开发工具，能进行一些普通的编程。MyEclipse 是商业级别的，Eclipse 安装插件后就可以成为 MyEclipse，但使用插件是要收费的。使用 MyEclipse 可以开发大型的 J2EE。

MyEclipse 是在 Eclipse 基础上的扩展。加了很多实用的插件！

MyEclipse 可以算是 Eclipse 的一个插件！

MyEclipse 比 Eclipse 多了很多功能！

MyEclipse 是收费的，而 Eclipse 是免费的！

1.5　Hello JSP

好了，我们的工具都已经到位了，现在要向 JSP 说 Hello 了！

(1) 打开 MyEclipse，如图 1-27 所示选择工作区，出现如图 1-28 所示的 MyEclipse 工作面板。

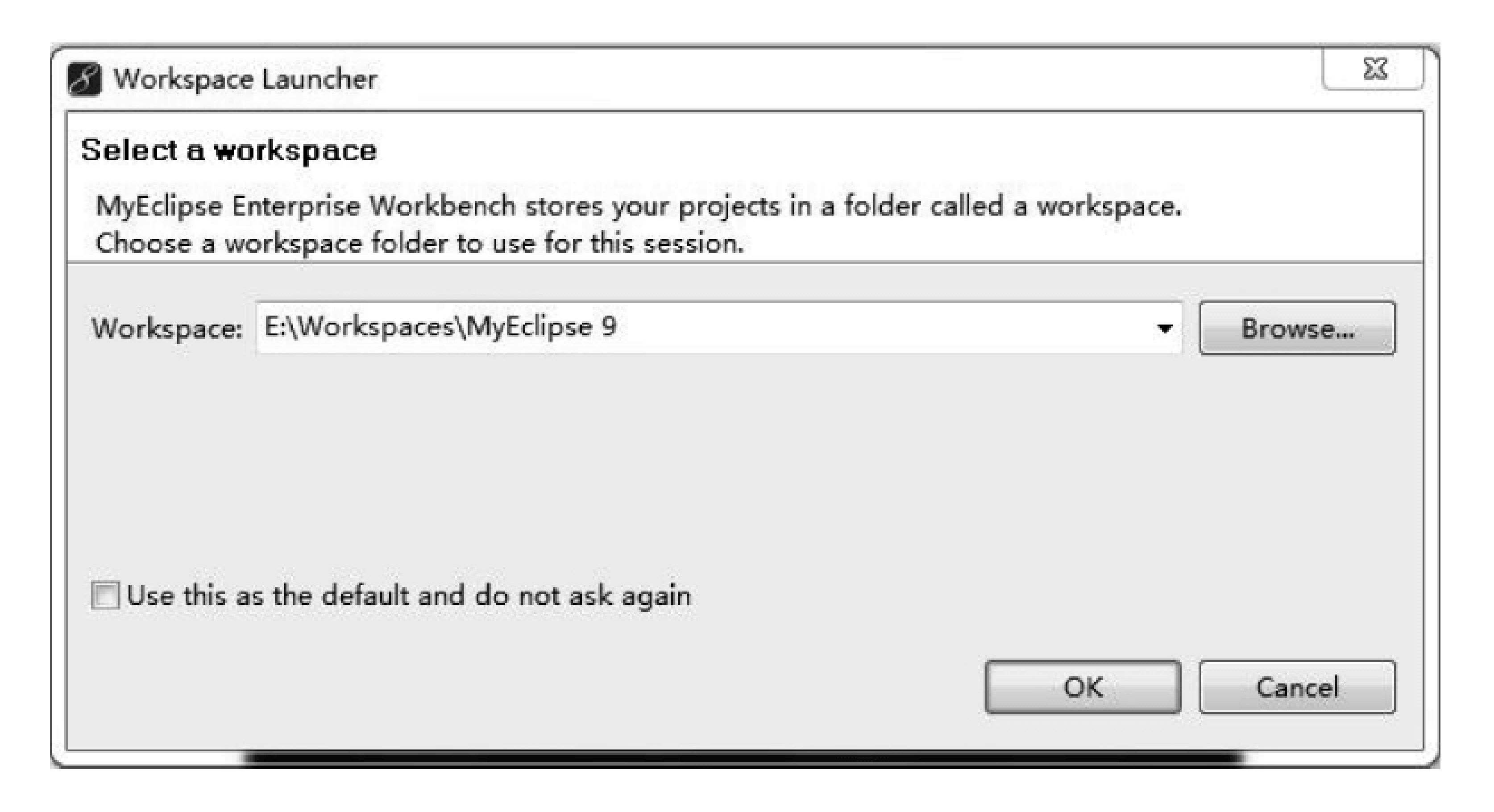


图 1-27 MyEclipse 选择工作区

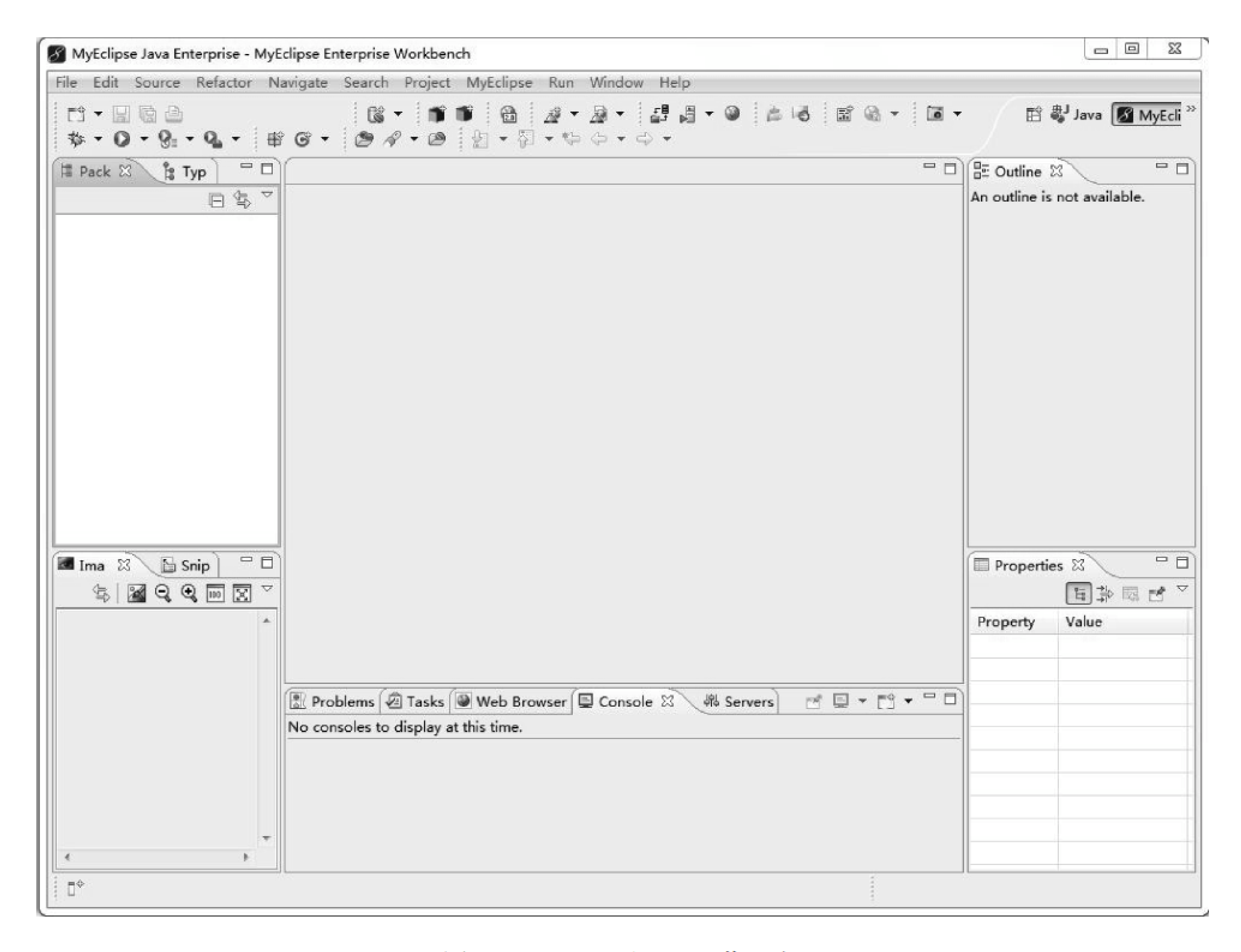


图 1-28 MyEclipse 工作面板

(2) 创建一个 Hello 的 Web 项目。选择菜单 File→New→Web Project,可以启动创建 Web 项目的向导,如图 1-29 所示。在这个图的 Project Name 中输入 Hello,然后选中 J2EE Specification Level 下面的 Java EE 5.0 单选钮,最后单击“Finish”按钮就可以创建 Web 项目了。



注意

选择哪个版本的 J2EE Specification Level 取决于你使用的服务器,例 Tomcat4,

Weblogic 9 以下版本请选择 J2EE 1.4，而 Tomcat 5，JBoss 4，或者 GlassFish 这样的服务器可以选择 Java EE 5.0。Java EE 5.0 可以直接使用 EL 表达式和 JSTL。

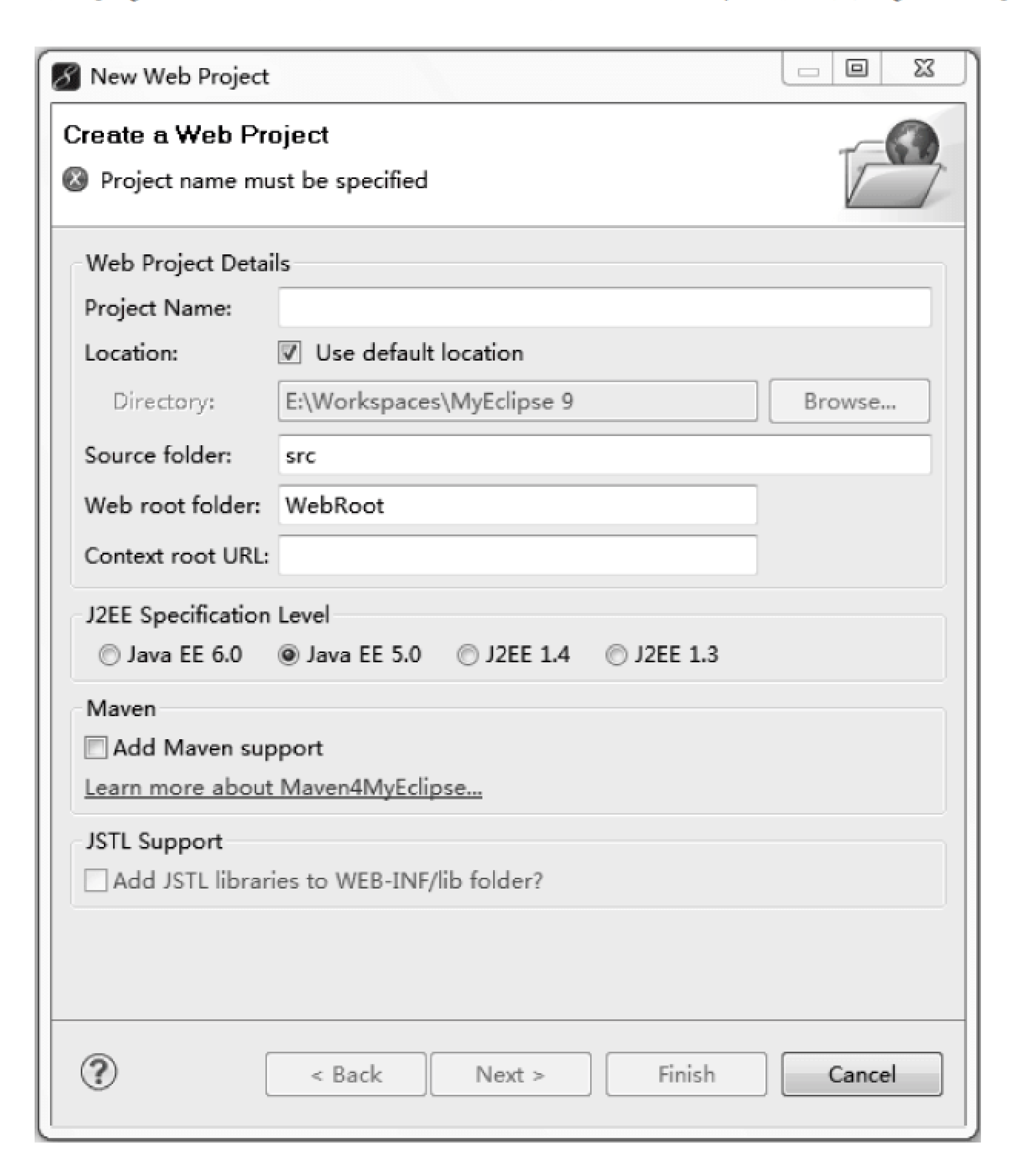


图 1-29　New Web Project 对话框

关于输入框的详细意义请参考表 1-2。

表 1-2　New Web Project 输入框的详细意义

选　项	描　　述
Project name	项目的名称。必须是有效的 Eclipse Java 项目名
Location	选中这个复选框来选择新项目的文件将存放到计算机中的其他位置
Directory	项目的默认存放位置是在 MyEclipse 启动时工作区目录下。当然可以选择位于工作区目录外的其他路径。注意：不能选择位于工作区子目录下的另一子目录，因为 Eclipse 禁止这种做法！
Source folder	Java 源代码目录——将包含 Java 包和.java 文件。这些内容会被加入到项目的 Java 构造路径中
Web root folder	这个目录将包含 Web 应用的内容、WEB-INF 目录以及对应的子目录。如果这个输入框内容为空，那么项目的根目录（"/"）将会成为 Web 根目录
Context root URL	MyEclipse 将 Web 项目部署到 Tomcat 时使用这个路径。默认使用的值是项目的名字。什么是上下文根目录？它是访问发布后的应用时所用的根路径，例如输入 Hello 后，将用地址 http://localhost:8080/Hello 来访问这个项目
J2EE specification	指定 J2EE 规范的版本。需要检查服务器的文档来了解其所支持的版本
Add JSTL 1.0 libraries	启用此选项来添加 Java Standard Template Library（Java 标准模版库 1.0 或者 1.1 版本）的 JAR 文件到新项目的<web-root>/WEB-INF/lib 目录下

创建后的 Hello 项目如图 1-30 所示。

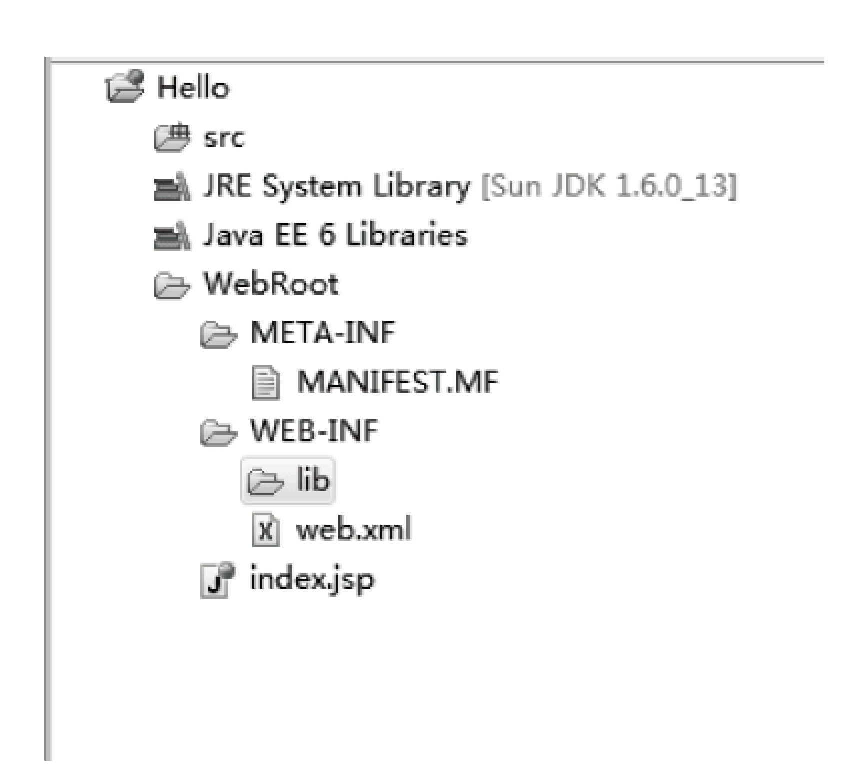


图 1-30 Hello Web 项目结构图

Web 项目是由多种资源文件构成的,包括编写的 Java 类、JSP 页面、各种静态资源以及发布的描述文件等。这些文件在 Web 应用目录的存放都是有一定限制和规定的。src 中存放的是 java 源文件(*.java),当然可以在 src 下创建包,例如 com.bean、com.dao 用来存放不同用途的 java 文件。WebRoot 是访问项目的根目录,表 1-3 中列出了 Web 应用目录结构的内容,并进行了相应的说明。

表 1-3 Web 项目的目录结构

目 录	说 明
/	Web 项目的根目录,该目录下的所有文件对客户端都可以访问,包括 JSP、HTML
/META-INF	用于存储包和扩展的配置数据,如安全性和版本信息
/WEB-INF	存放应用程序所使用的各种资源,该目录及其子目录对客户端都是不可以访问的,其中包括 web.xml(部署表述符)
/WEB-INF/classes	存放应用的所有 class 文件
/WEB-INF/lib	存放 Web 应用使用的 JAR 文件

web.xml 是 Web 项目的一个核心文件,它必须在 WEB-INF 目录下,该文件控制整个项目的行为方式和方法,打开 web.xml 文件,其内容如示例 1-2 所示。

示例 1-2

```
/**
 * web.xml
 */
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
     xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
<welcome-file-list>
```

```
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

“<? xml version="1.0" encoding="UTF-8"? >”是头声明，它指出使用的 XML 版本并给出文件的字符编码。DOCYTPE 声明必须立即出现在此头声明之后。这个声明告诉服务器适用的 servlet 规范的版本，并指定管理此文件其余部分内容的语法的 DTD(Document Type Definition，文档类型定义)。所有部署描述符文件的顶层(根)元素为 web-app。而

```
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

则设置了欢迎页面 index.jsp，即此时在浏览器中输入的访问地址不再需要输入具体的资源文件，而是直接访问 Web 应用名称即可(http://localhost:8080/Hello)，此时 Tomcat 会自动读取在 web.xml 文件的配置信息，然后在浏览器中显示欢迎页面 index.jsp。当然 web.xml 还有其他配置，在以后的学习中注意总结积累。


经验

在 Tomcat 的运行过程中，Tomcat 类加载器会首先加载 classes 目录下的 class 文件，然后再加载 lib 目录下的类。需要注意的是，如果在两个目录下存在同名的类，那么 classes 目录下的类具有优先权。

(3) 在 WebRoot 下新建 JSP 文件 HelloWorld.jsp，如图 1-31 所示。

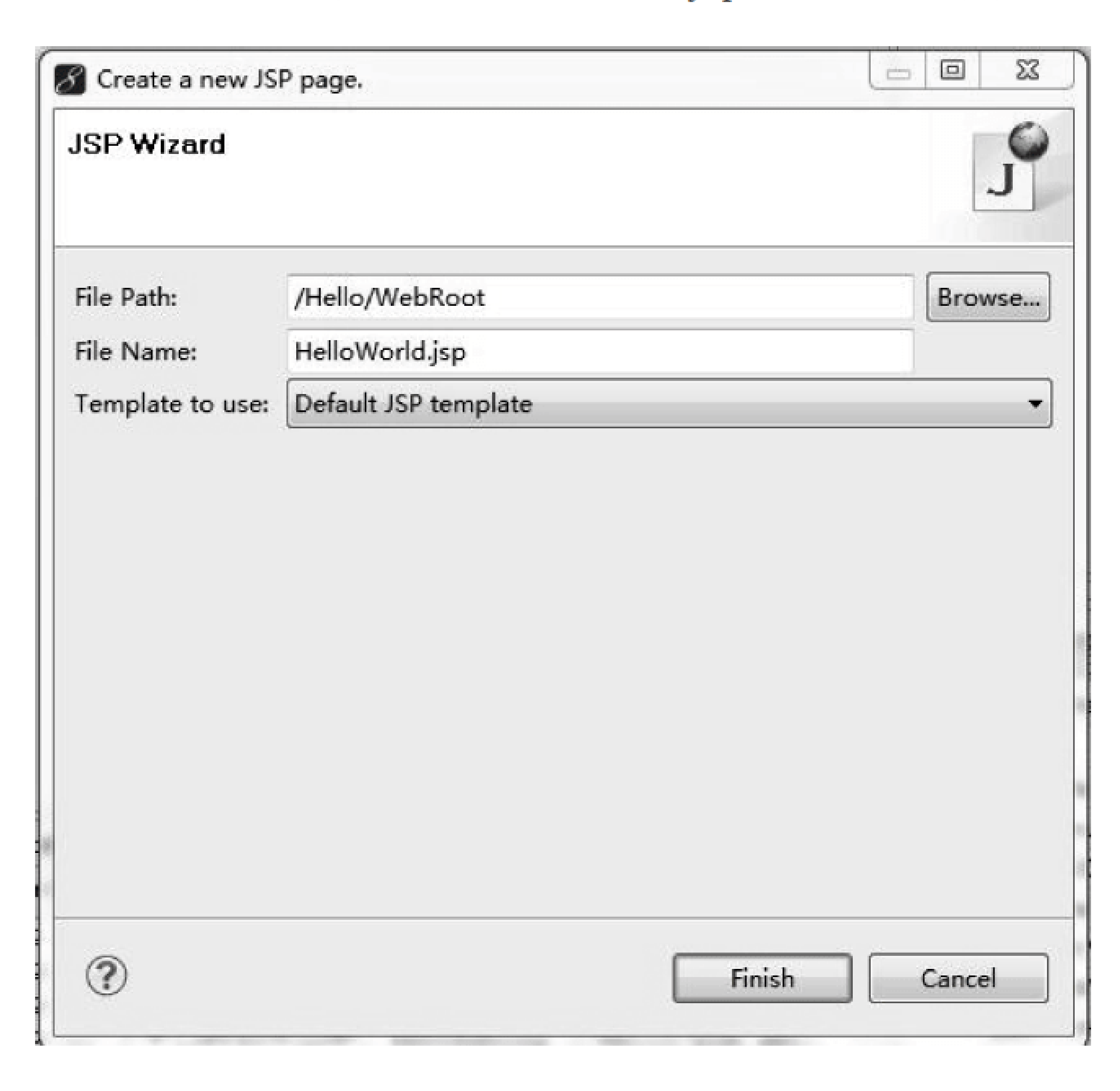


图 1-31　创建 HelloWorld.jsp 页面

(4) 双击 HelloWorld.jsp 打开此文件,如图 1-32 所示。

HelloWorld.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="ISO-8859-1"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>My JSP 'HelloWorld.jsp' starting page</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
    This is my JSP page. <br>
  </body>
</html>
```

图 1-32 HelloWorld.jsp 代码

(5) 修改代码,图 1-33 为 HelloWorld.jsp 修改后的代码。

HelloWorld.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>My JSP 'HelloWorld.jsp' starting page</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
    Hello World!! I am  JSP!! <br>
  </body>
</html>
```

图 1-33 HelloWorld.jsp 修改后代码

(6) 单击工具条中的 ,Delply MyEclipse J2EE project to Server…部署项目,出现如图 1-34 所示的界面。单击"Add"按钮,出现如图 1-35 所示的界面。

这时的 Server 是 MyEclipse 提供的 Server,我们需要自己配置,单击"Edit server connectors"选择条。结果如图 1-36 所示。

我们已经安装了 Tomcat 7,所以在这里配置 Tomcat 7,单击 Tomcat 7.x。

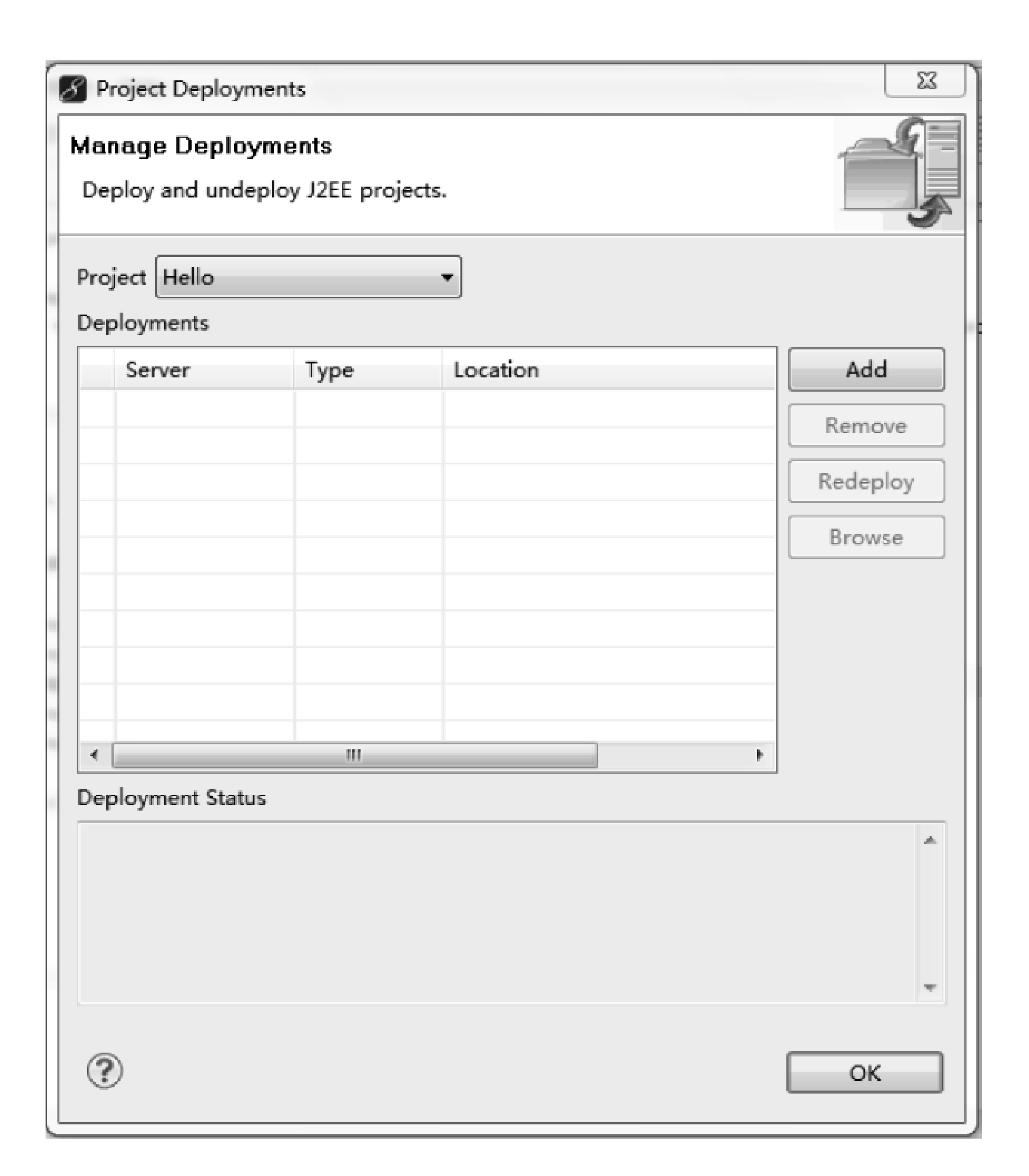


图 1-34　部署项目

New Deployment

New Deployment

Select Application server for deployment

Web Project: Hello

Server:

Edit server connectors...

Deploy type: Exploded Archive (development mode) Packaged Archive (production mode)

Deploy Location:

Finish Cancel

图 1-35　部署添加项目

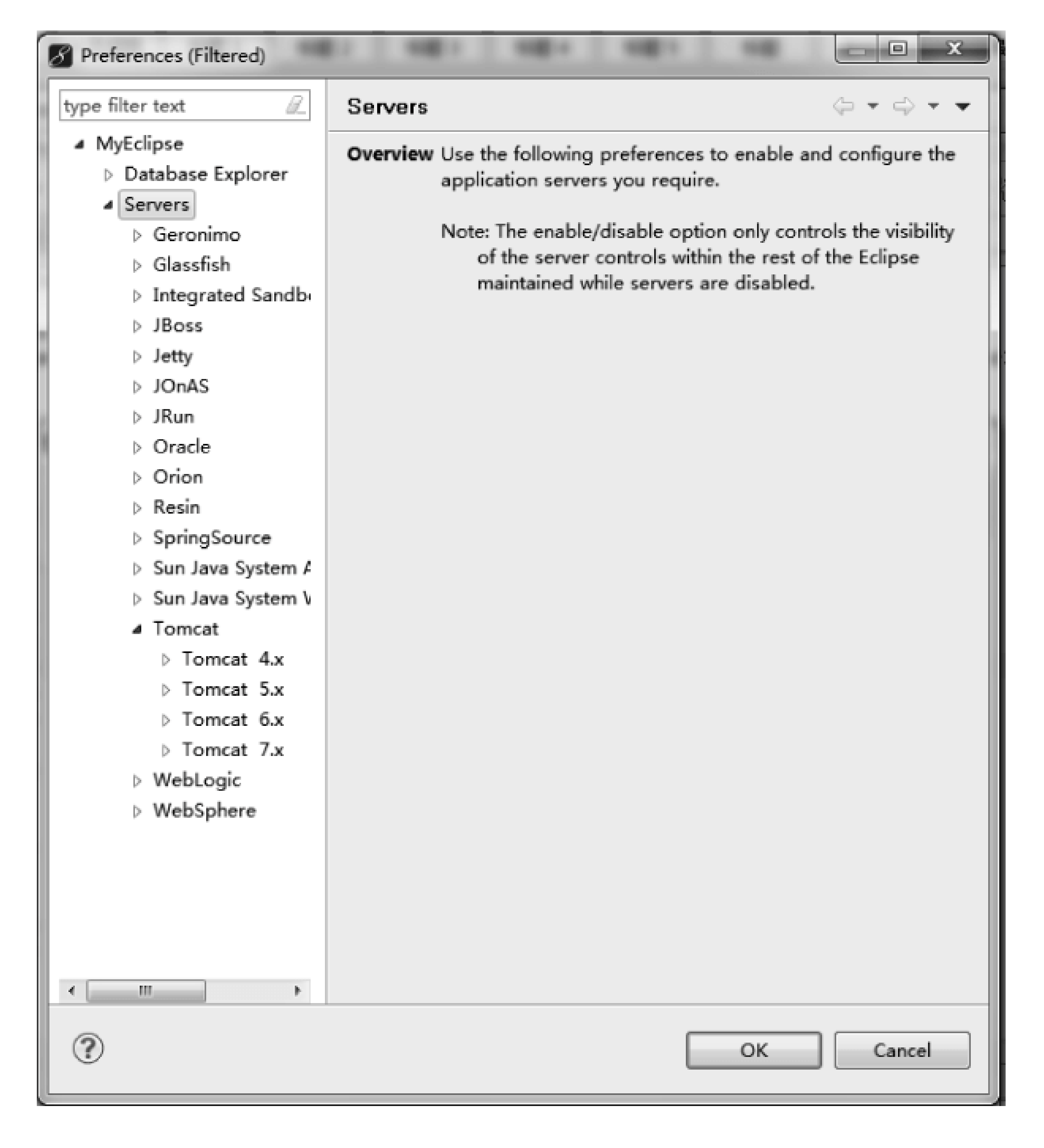


图 1-36　配置 server

在图 1-37 中，Tomcat 7. x 选中 Enable 单选按钮，在 Tomcat home directory 后单击“Browser”按钮。选择安装 Tomcat 的根路径，然后单击“Apply”按钮。

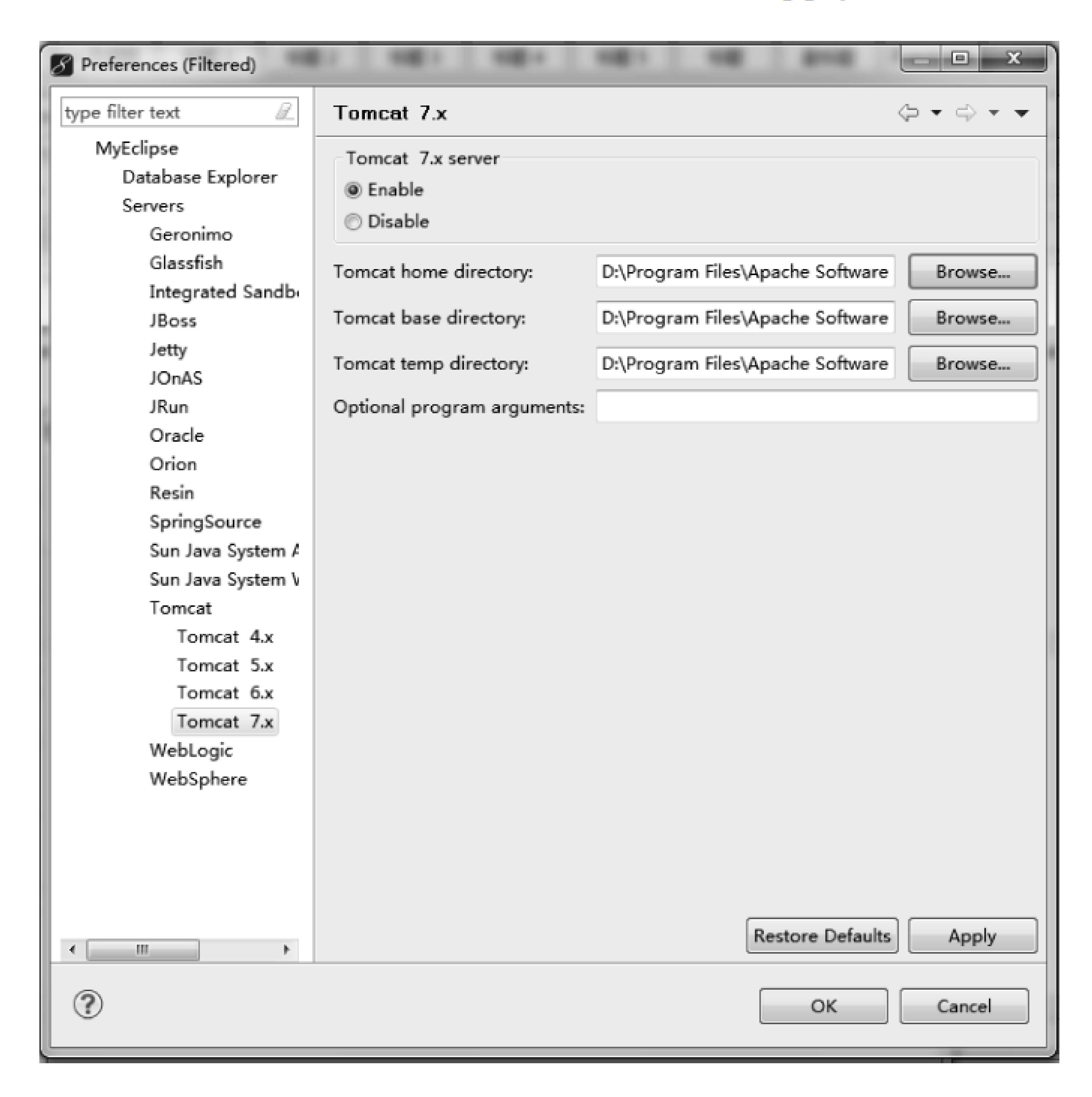


图 1-37　配置 Tomcat7. x



接下来选择 Tomcat 7. x 的 JDK，如图 1-38 所示。

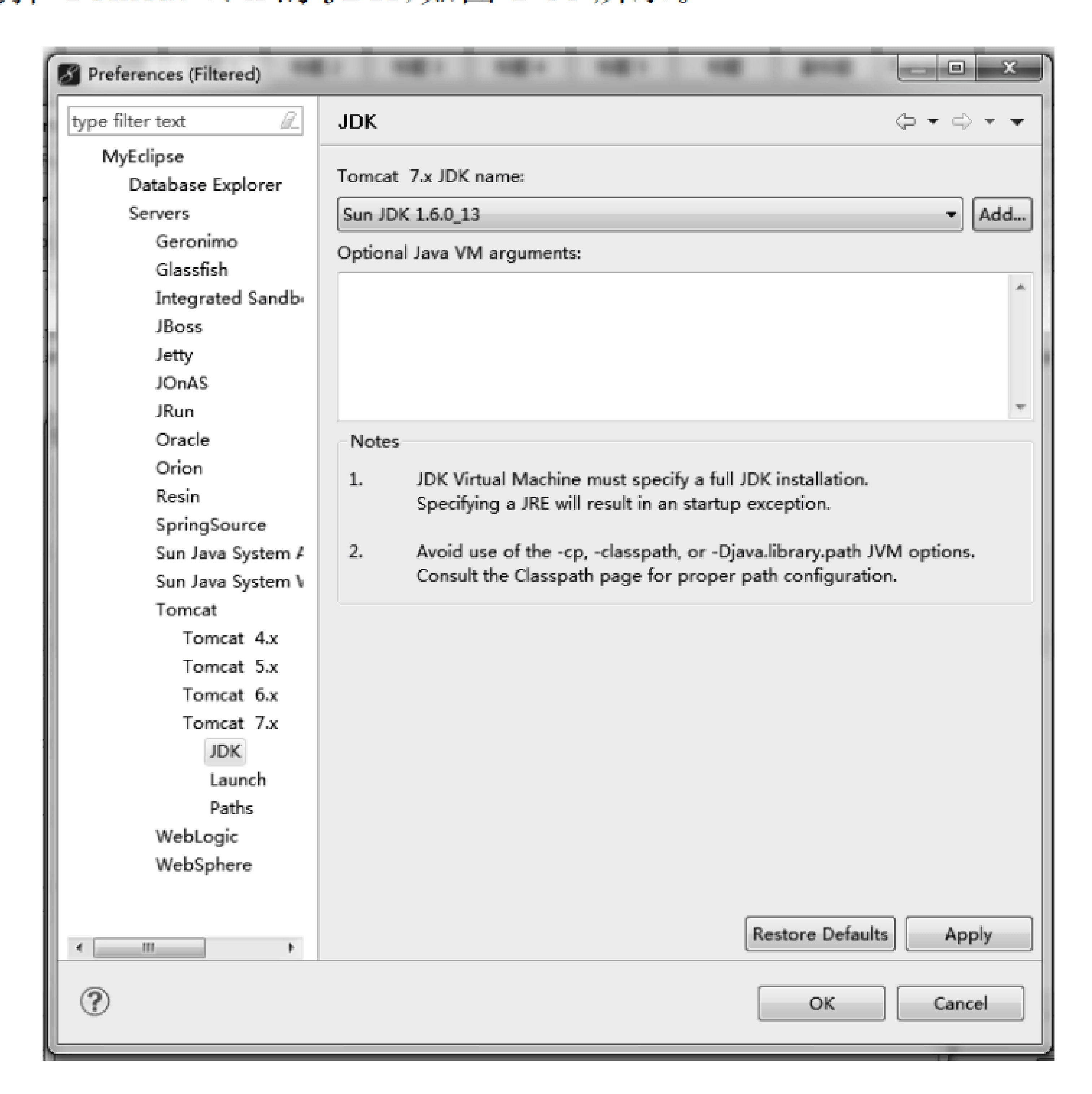


图 1-38　配置 Tomcat7 的 JDK

单击“Add”按钮，选择安装 JRE 的根路径，如图 1-39 所示，然后单击“Finish”按钮，MyEclipse 中的 Tomcat 配置完毕，以后直接使用即可，(在同一个 workspace 中)无需再次配置。

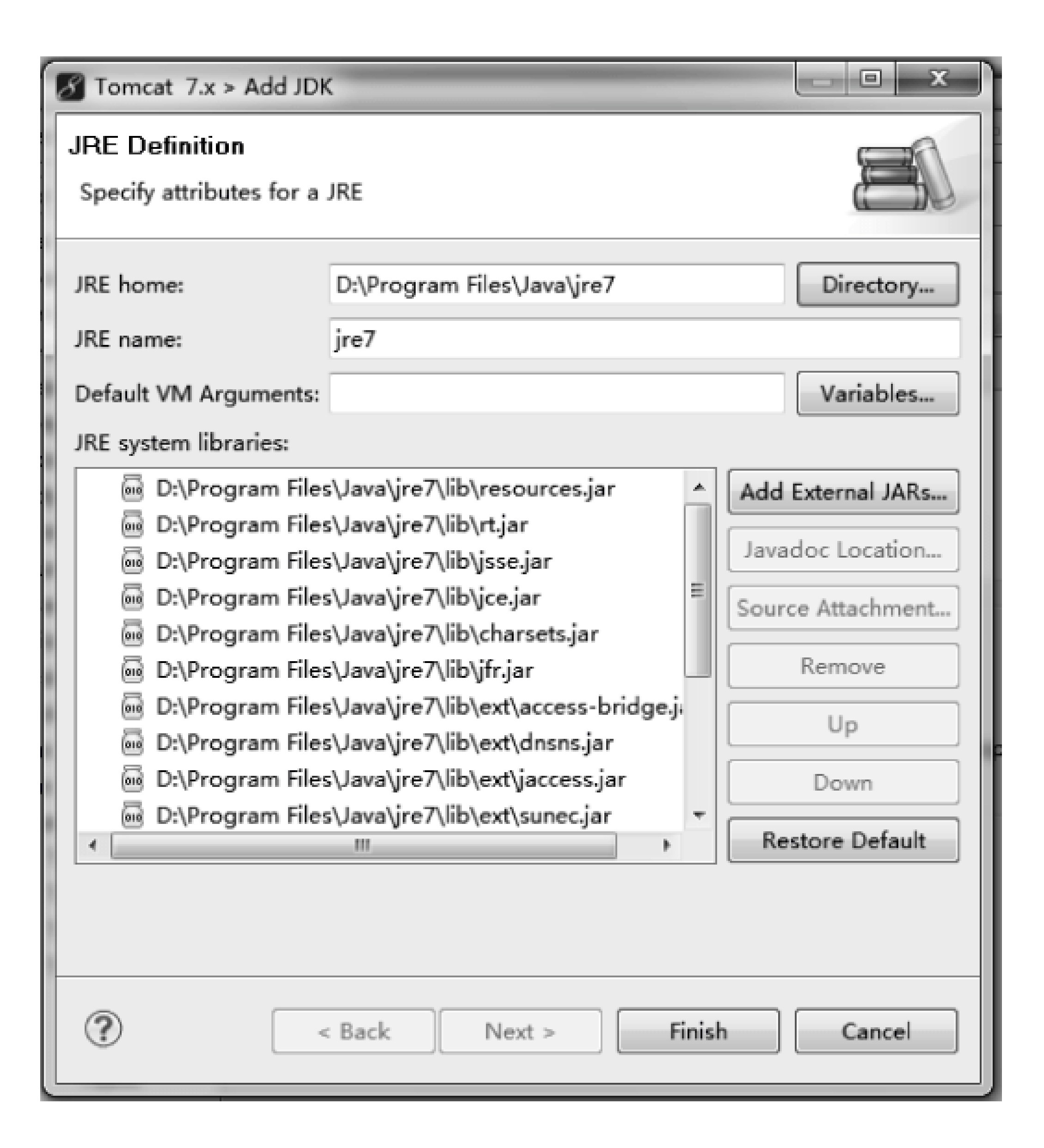


图 1-39　配置 Tomcat7 JDK

重新部署该项目，如图 1-40 所示，在 server 中即出现 Tomcat 7. x(刚配置好的)，选择 Tomcat 7. x，单击“Finish”按钮，完成部署。

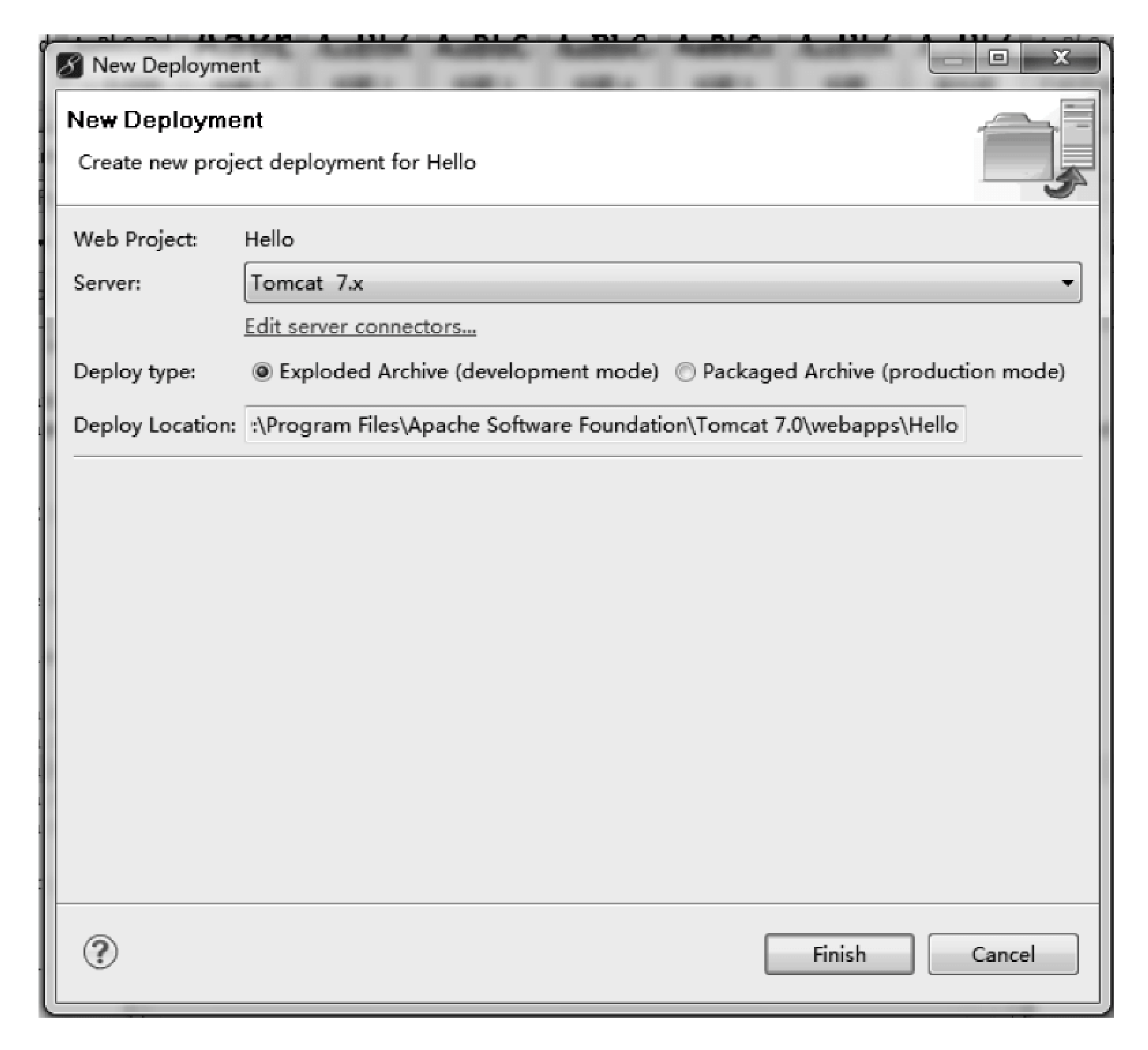


图 1-40 部署项目

(7) 好了，现在即可在浏览器中看到我们的成果了，单击 IE 浏览器，在地址栏中输入 http://localhost:8080/Hello/HelloWorld. jsp，显示如图 1-41 所示。

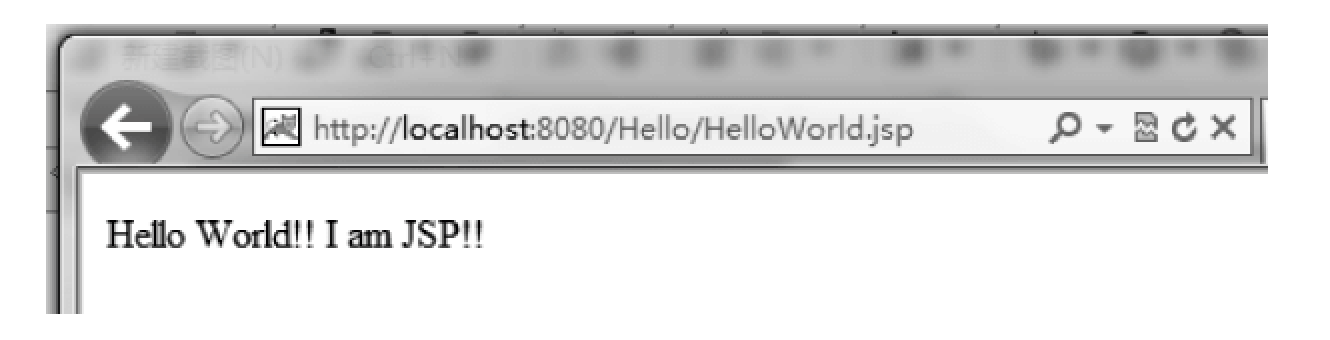


图 1-41 HelloWorld. jsp 页面

1.6 上机练习

1. 安装 JDK。

需求说明：请下载、安装 JDK，并配置环境变量。

实现思路：参照 1.4.1 节。

2. 安装 Tomcat。

需求说明：请下载、安装 Tomcat，并测试安装是否成功。

实现思路：参照 1.4.2 节。

3. 安装 MyEclipse。

需求说明：请下载、安装 MyEclipse。

实现思路：参照 1.4.3 节。

4. 编写你的第一个 JSP 页面 Hello.jsp。

需求说明：通过前 3 个上机练习，已经把开发 JSP 的环境搭建起来，接下来就要利用 MyEclipse，创建一个 Web 项目，编写 Hello.jsp，并在 IE 浏览器中显示内容。

实现思路：参照 1.5 节。

1.7 总 结

(1) 静态网页是指纯粹 HTML 代码格式的网页。动态网页是指其页面信息可以根据需求或者用户的浏览状况，实现与用户交流和页面信息自动更新的网页。

(2) B/S 架构统一了客户端，将系统功能实现的核心部分集中到服务器上，通过服务器同数据库服务器进行通信，简化了系统的开发、维护和使用。

(3) URL(Uniform Resource Locator)即统一资源定位符，也称为网页地址，是互联网上标准的资源地址。

URL 的组成部分如下：

- 第一部分：协议；
- 第二部分：主机名(有时包含端口号)；
- 第三部分：路径。

(4) 搭建 JSP 开发环境：

- 安装 JDK；
- 安装 Tomcat；
- 安装 MyEclipse；
- 运行 Web 项目。

(5) 开发 JSP 动态网站的步骤如下：

- 创建一个 Web 项目；
- 设计 Web 项目的目录结构；
- 编写 Web 项目的代码；
- 部署 Web 项目；
- 运行 Web 项目。

1.8 作 业

一、选择题

1. 以下选项中(　　)是正确的 URL。

A. http://www.sina.com.cn　　　　B. www.baidu.com

C. ftp:ftp.link.com　　　　D. yahoo.com.cn/news/index.html

2. 在静态网页中，下面说法错误的是(　　)。

A. 在静态网页中可以插入 GIF 动画图片　　B. 在静态网页中可以插入 AVI 动画
C. 在静态网页中可以插入 Java 代码片断　　D. 在静态网页中可以插入 Flash 动画

3. 在设计 Web 项目的目录结构时，一般把 JSP 和 HTML 文件放在(　　)下。

A. src 目录　　B. 文档根目录或其子文件夹
C. META-INF 目录　　D. WEB-INF 目录

4. 在 Web 项目的目录结构中 web.xml 文件位于(　　)。

A. src 目录中　　B. 文档根目录
C. META-INF 目录　　D. WEB-INF 目录

二、简答题

1. 什么是静态网页？什么是动态网页？两者的区别是什么？
2. 什么是 B/S 模式？
3. 如何配置 JSP 开发环境？

第 2 章　静态网页开发基础

本章学习目标

- 熟练掌握 HTML 的常用标签
- 熟练使用 HTML 设计基本网页
- 熟练使用 CSS 样式的三种方式
- 理解并会使用 CSS 的四类选择器
- 掌握 JavaScript 的简单应用

在第 1 章中，我们学习如何搭建 JSP 的开发运行环境，并且创建了一个 Web 项目。接下来为了制作 JSP 网页，还需要学习相关网页的基础知识，HTML＋CSS＋JavaScript 是制作静态网页的基本模式。本章就来介绍这些基础知识。

2.1　HTML 基础

什么是 HTML 呢？HTML(Hyper Text Markup Language)被称为超文本标签语言，是用于描述网页文档的一种标记语言。它包含很多标签(例如：<p>段落，<h1>标题)，告诉浏览器如何显示页面，其主要特点如下：

(1) 简易性：各类 HTML 标签简单易学，便于制作、开发；

(2) 平台无关性：这是 HTML 语言的最大优点，它包括“硬件”平台无关性和“软件”平台无关性，即不论是普通的 PC 机，还是专业的 MAC，操作系统不论是 Windows，还是 Linux，HTML 文档都可以得到广泛的应用和传输。

HTML 是制作静态网页的核心，JSP 的开发当然离不开 HTML。下面就来介绍 HTML 文档的基本结构和各类标签的用法。

2.1.1　基本结构

如图 2-1 所示，HTML 的基本结构分为两部分：头部分和主体部分。

其中<html>…</html>、<head>…</head>、<body>…</body>都是标签。标签由一对尖括号及其标签符组成，没有反斜杠“/”的是开始标签，有反斜杠的是结束标签。每个 HTML 文件是以<html>标签开头，而以</html>标签结束。

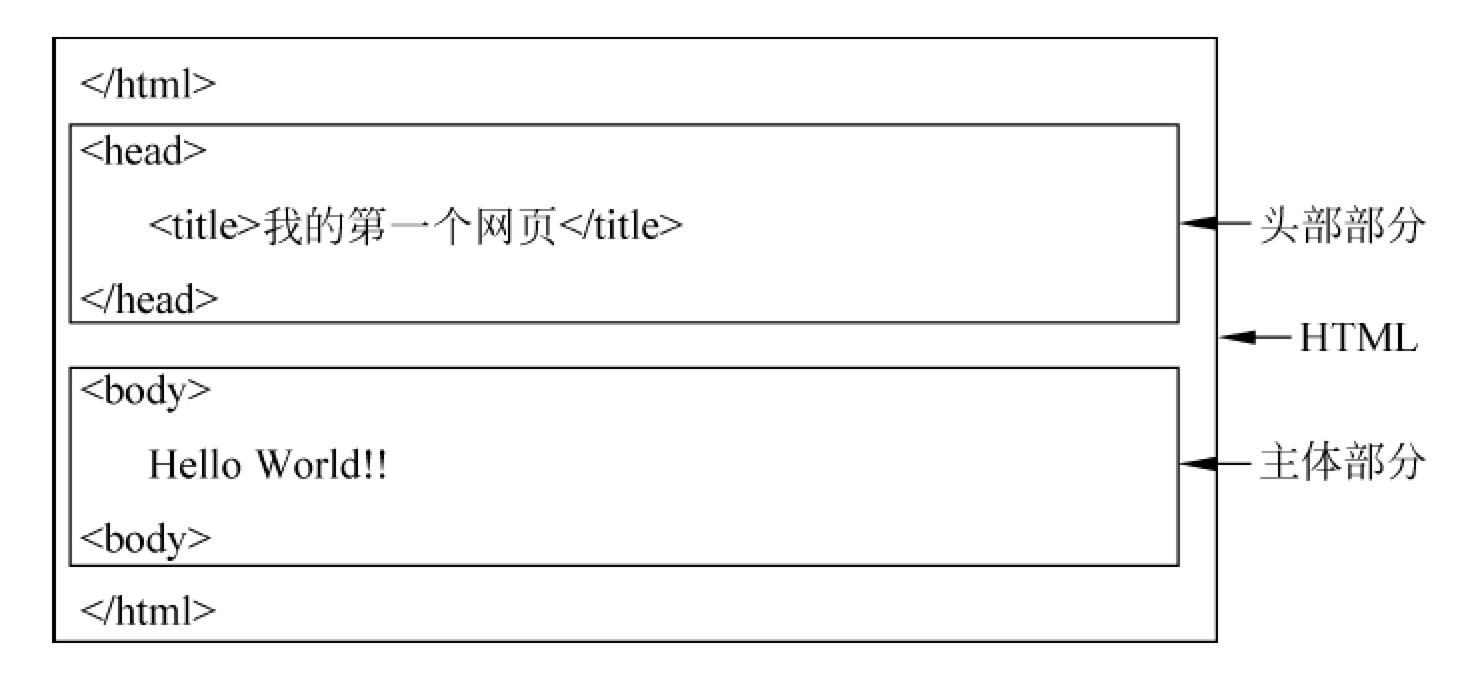


图 2-1　HTML 基本结构



说 明

✍ HTML 区分大小写吗？

在 HTML 4.0 及以前的版本中，W3C 标准是不区分大小写的，但是在 HTML 5.0 版本后，W3C 明确规定，标签必须用小写格式，所以，大家在编写 HTML 时尽量养成小写习惯。

制作 HTML 文件有很多的专用网页制作工具，比如 Dreamweaver、FrontPage 等。那么如果没有这些开发工具，我们用 Windows 操作系统自带的记事本软件也可以编写 HTML 网页。但不管使用哪个编辑工具，HTML 网页最后都要保存为扩展名是“html”或者“htm”的纯文本文件。现在我们就来用记事本软件编写一个简单的 HTML 网页。

在 Windows 中打开记事本程序。

(1) 在记事本中输入 HTML 代码，如图 2-2 所示；

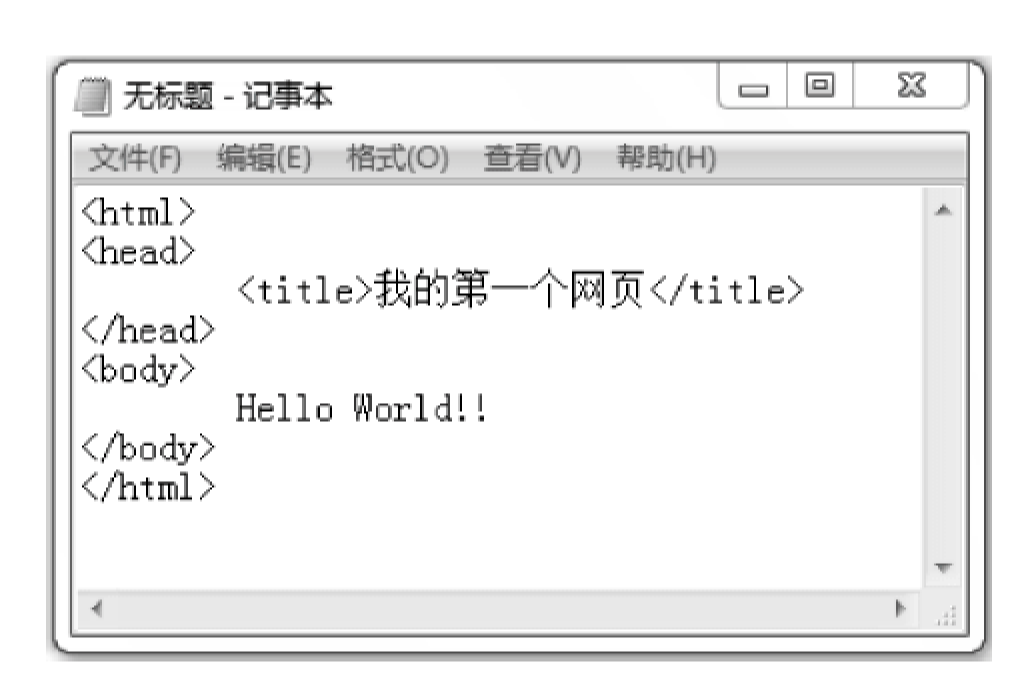


图 2-2　在记事本中编辑 HTML

(2) 写完代码后保存在名为“hello. html”的文件中；

(3) Windows 自动调用浏览器软件(IE)打开 HTML 文档，如图 2-3 所示，< title >标签定义文档的标题，即浏览器工具栏中显示的标题。



图 2-3　我的第一个网页

2.1.2 常用标签

这里只介绍几个常用的标签，其他的标签可参考静态网页设计的相关书籍或 www.w3school.com.cn/html 网站。

1. 文本标签

文本标签在 Web 页面中被用来设置文本内容的布局和格式，表 2-1 列出的是常用的 HTML 文本标签。

表 2-1 HTML 文本标签表

开始标签	结束标签	含　　义
<p>	</p>	对文本进行分段
<div>	</div>	对文本进行分块
 	无	换行，开始新的一行
<hr>	无	水平线
		用于设置文本的字体和颜色
		粗体文本
<i>	</i>	斜体文本
<h1-h6>	</h1-h6>	各级标题

示例2－1

```
/**
 * HTML 文本标签演示代码
 */
<html>
<head>
<title>HTML 文本标签演示代码</title>
</head>
<body>
    <h1>见或不见</h1>
    <hr size="3" width="50%" align="center">
    <h2><i>你见，或者不见我</i></h2>
    <h3>我就在那里</h3>
    <h4>不悲不喜</h4>
    <h5>你念，或者不念我</h5>
    <h6>情就在那里</h6>
    <i>不来不去</i>
    <br/>
    <font face="隶书"color="red">你爱，或者不爱我</font>
    <b>爱就在那里</b>
    不增不减<br/>
    <p align="center">你跟，或者不跟我<br/>我的手就在你手里<br/>不舍不弃<br /><p>
    <div align="right">来我的怀里<br/>或者<br/>让我住进你的心里<br/>默然 相爱<br/>寂
静 欢喜</div>
</body>
</html>
```

在有的标签中还可以加一些属性参数，例如< hr size＝"3" width＝"50%" align＝"center">，指的是画一条高度为“3”，宽度为页面“50%”，居中对齐的线，size、width、align 为标签< hr >的属性，"3"、"50%" 、"center"为属性对应的属性值；< font face＝"隶书"color＝"red">，要求字体为“隶书”，颜色为红色的。标签也可以嵌套使用，例如< h2 > < i >你见，或者不见我</i > </h2 >，就把< h2 >与< i >嵌套使用，使得嵌套内的字既是标题二又是斜体的，这里要注意嵌套标签的结束顺序。有的标签带自动换行功能（例如< h1-6 >、< p >等），有的则不带（例如< font >、< i >等）。此示例的效果如图 2-4 所示。

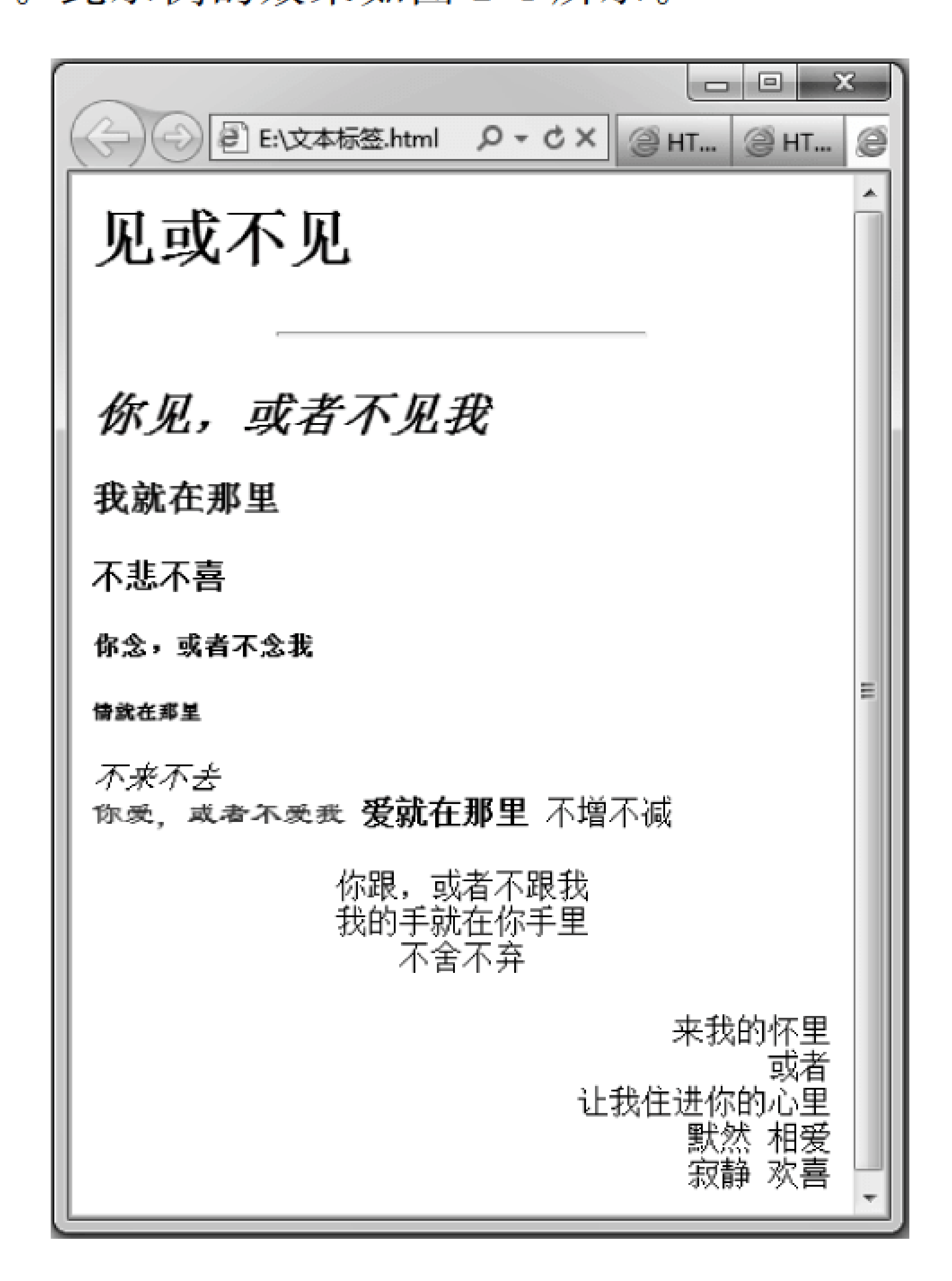


图 2-4　示例 2-1 的效果图

2. 列表标签

列表标签被用来将 Web 内容组织成为未排序列表、排序列表、自定义列表。表 2-2 列出了常用的 HTML 列表标签。

表 2-2　HTML 列表标签表

开始标签	结束标签	含　　义
< ul >	</ ul >	未排序（符号）列表
< ol >	</ ol >	排序（编号）列表
< li >	</ li >	列表中的项目

示例2－2

```
/* *
 * HTML 列表标签演示代码
 * /
```

```
<html>
<head>
<title>HTML 嵌套列表标签演示代码</title>
</head>
<body>
    <h4>一个嵌套列表:</h4>
    <ol type="1"  start="2">
        <li>咖啡</li>
        <li>茶
            <ul type="circle">
                <li>红茶</li>
                <li>绿茶</li>
            </ul>
        </li>
        <li>牛奶</li>
    </ol>
</body>
</html>
```

未排序列表标签<ul>常用的属性参数：type 设定的符号样式(可选值：disc，circle(默认)，square，分别对应于实心圆、空心圆和正方形)。排序列表标签<ol>常用的属性参数：type 设定编号的样式(可选值：1(默认值)，a，A，I，i，分别对应于正整数、小写英文字母、大写英文字母、大写罗马数字、小写罗马数字)，start 设定开始的编号。当然列表标签也可以嵌套使用。效果如图 2-5 所示。

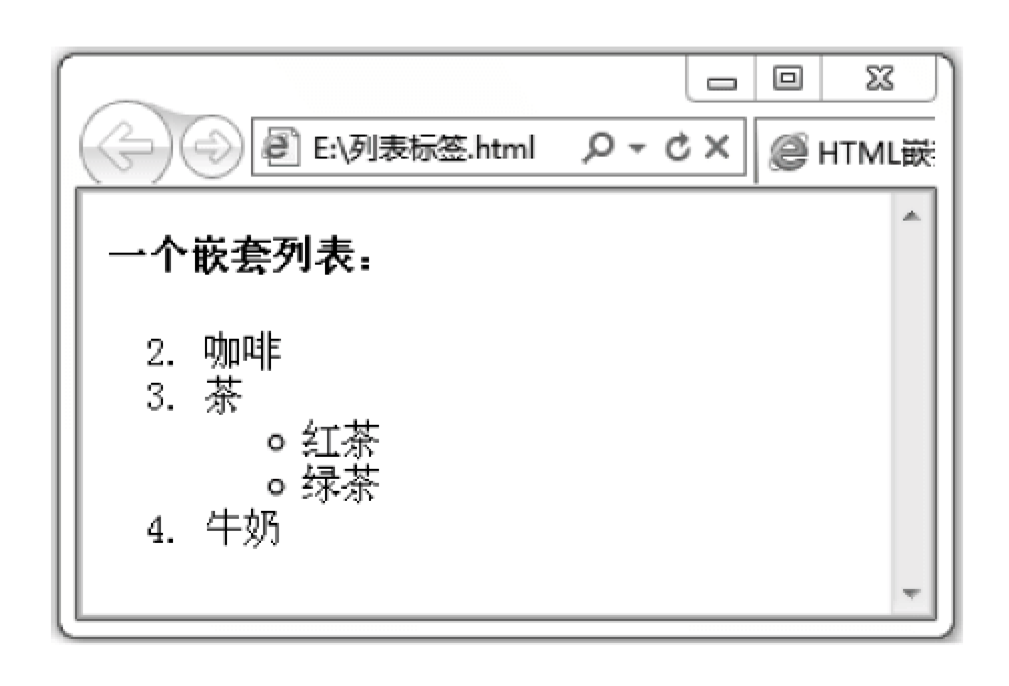


图 2-5　示例 2-2 的效果图

3. 表格标签

表格标签可以用来以表格的方式组织需要显示的内容。表 2-3 列出常用的 HTML 表格标签。

表 2-3　HTML 表格标签表

开始标签	结束标签	含　义
<table>	</table>	表格
<tr>	</tr>	表格中的行
<td>	</td>	表格中的单元格

续表

开始标签	结束标签	含　　义
<th>	</th>	表头
<caption>	</caption>	表格标题

示例2-3

```
/**
 *HTML 表格标签演示代码
 */
<html>
<head>
<title>HTML 表格标签演示代码</title>
</head>
<body>
<table width="200" border="1" cellpadding="2" cellspacing="3">
    <caption>商品信息</caption>
    <tr>
        <th>商品名称</th>
        <th>数量</th>
        <th>备注</th>
    </tr>
    <tr align="right" bgcolor="#33FFFF">
        <td>雨伞</td>
        <td>100</td>
        <td rowspan="2">合并列</td>
    </tr>
    <tr>
        <td align="center" valign="middle">皮鞋</td>
        <td>200</td>
    </tr>
    <tr>
        <td colspan="3">合并行</td>
    </tr>
</table>
</body>
</html>
```

(1) 表格标签<table>常用的属性参数

width　设定表格宽度，接受绝对值（如 80，单位是像素）及相对值（如 80%，整个 HTML 页面宽度的 80%）；

border　设定单元格边线的宽度，以像素为单位。包括每个单元格的边线以及整个表格的边界线。border 属性默认为 0，在这种情况下表格线将被隐藏起来；

cellspacing　用于设置表格的单元格之间的距离，单位为像素，默认值为 2px；

cellpadding　用于设置单元格内容与边线之间的距离，单位为像素，默认值为 2px；

align　用于设置表格摆放的水平位置(可选值为 left,center 和 right,即左对齐、居中和右对齐,默认为 left);

valign　用于设置表格摆放的垂直位置(可选值为:top,center 和 bottom,即上对齐、居中和下对齐,默认为 center);

background　用于设定表格的背景图片,属性值为指向一幅图片的 VRL 值;

bgcolor　设定表格的背景颜色(与 background 不能同用);

bordercolor　设定表格的边框颜色。

(2) 表格行标签< tr >常用的属性参数

align　设定表格一行内文字、图等摆放的水平位置(可选值为:left,right 和 center),默认值为 letf;

valign　设定表格一行内文字、图等摆放的垂直位置(可选值为:top,middle 和 bottom),默认值为 middle;

bgcolor　设定表格一行的背景颜色。

(3) 表格单元格标签< td >,常用属性参数

width　设定单元格的宽度,接受绝对值及相对值;

height　设定单元格的高度;

colspan　设定水平跨越的单元格数,默认值为 1;

rowspan　设定垂直跨越的单元格数,默认值为 1;

align　设定该单元格内文字、图等摆放的水平位置(可选值为:left,center 和 right),默认值为 left;

valign　设定该单元格内文字、图等摆放的垂直位置(可选值为:top,middle 和 bottom),默认值为 middle;

bgcolor　设定该单元格的背景颜色。

(4) 表头标签< th >

作用与< td >相近,表示一个单元格,不同的是< th >表示单元格中的文字以粗体出现。

(5) 表格标题标签< caption >

为表格定义一个标题行,用于存放该表格的标题。常用的属性参数:

align　设定标题相对于表格的水平位置(可选值为:left,center 和 right),默认值为 left;

valign　设定标题相对于表格的垂直位置(可选值为:top,middle 和 bottom),默认值为 middle。

示例 2-3 的效果如图 2-6 所示。

图 2-6　示例 2-3 的效果图

4. 表单标签

HTML 页面上的表单标签用于向用户提供一些图像控件,例如单行文本框、多行文本框、密码框,目的是让用户输入数据,并将用户输入的数据进行提交。表 2-4 是常用的 HTML 表单标签。

表 2-4 HTML 表单标签表

开始标签	结束标签	含 义
<form>	</form>	表单
<input>	</input>	表单中的输入标签
<select>	</select>	表单中的选择列表标签
<option>	</option>	表单中的选择列表项标签
<textarea>	</textarea>	可滚动的多行文本框标签

示例2-4

```
/**
 * HTML 表单标签演示代码
 */
<html>
<head>
<title>HTML 文本标签演示代码</title>
</head>
<body>
<form action = "" method = "post" name = "myForm">
  <p>姓名:<input type = "text" name = "name" /></p>
  <p>密码:<input type = "password" name = "password" /></p>
  <p>爱好: 看书<input type = "checkbox" name = "book">听音乐:<input type = "checkbox" name
 = "music"></p>
  <p>性别:男<input type = "radio" checked = "checked" name = "Sex" value = "male" />女<input
type = "radio" name = "Sex" value = "female" /></p>
  <p>所在地:<select name = "place">
                <option value = "shandong">山东</option>
                <option value = "hunan" selected = "selected">湖南</option>
                <option value = "hebei">河北</option>
            </select></p>
  <p>个人简介:
    <textarea name = "textarea" cols = "20" rows = "5">
        这是一个害羞的人!没有介绍自己啊.
    </textarea>
  </p>
  <input type = "submit" value = "提交" /><input type = "reset" value = "重填" />
</form>
</body>
</html>
```

(1) 表单标签<form>常用属性参数:

name　指定表单的名字;

action　指定处理表单中数据的应用程序(本书中是 JSP 或 Servlet)的位置;

method　指定传输表单数据的方式(可选值为:post 和 get)。

(2) 输入标签<input>常用属性参数:

由于输入标签的属性参数 type 有很多选择,不同的选择表示不同的输入方式。例如:

text(单行文本框),radio(单选框),checkbox(复选框),password(密码框),submit/ reset(提交按钮/清除按钮),file(文件域),hidden(隐藏域),button(普通按钮)。不同的选择表示不同的输入方式,且其他属性参数亦因此而异。各种输入方式详细的属性参数设定在接下来的5小节将详细介绍。

(3) 选择列表标签<select>,用来表示表单中的选择列表与<option>一起使用,常用属性参数:

name　设定选择列表的名称,供应用程序做识别之用;

multiple　设定选择列表是否允许有多重选择。

(4) 选项标签<option>,用来表示选择列的选项,常用属性参数:

value　设定该选项的值;

selected　设定该选项被选中。

(5) 多行文本标签<textarea>常用属性参数:

name　设定多行文本控件的名称,供应用程序做识别之用;

cols　设定多行文本字段的宽度,即每行能容纳多少个英文字符;

rows　设定多行文本字段的高度,即显示多少行。

示例2-4的效果如图2-7所示。

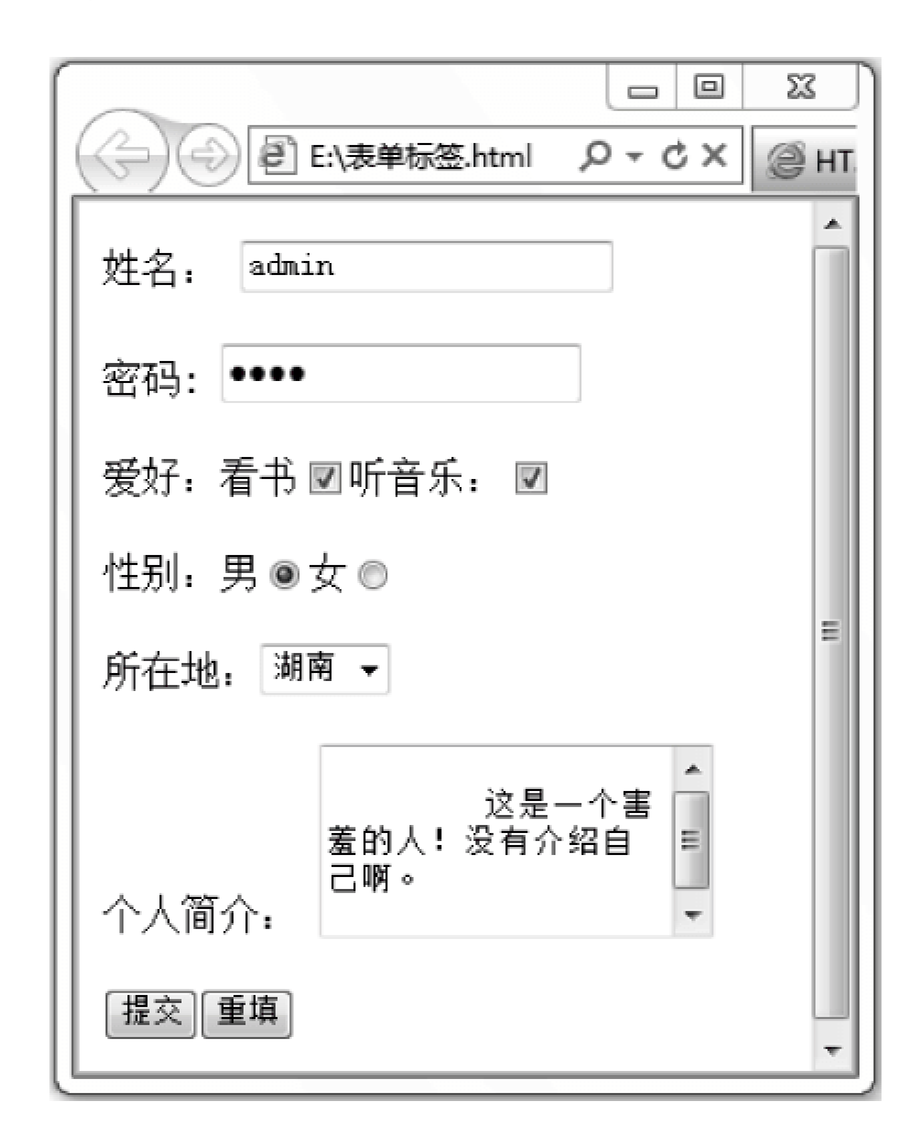


图2-7　示例2-4的效果图

5. <input>标签详解

<input>的属性参数type有很多的选择:text,radio,checkbox,password,submit/reset,file,hidden,button。不同的选择表示不同的输入方式,且其他属性参数亦因此而异。下面将详细介绍各种输入方式及其属性参数设定。

(1) 单行文本框(text)。

例如:<input type = "text" name = "userName" value = "zhangsan" align = "middle" size = "10" maxlength ="255">

- type= "text":输入方式为Text,能产生一个单行文本框;maxlength ="255"表示

用户最多能输入 255 个英文字符。

- name= " userName ":设定该文本框的名称。这是最重要的一个,应用程序就是通过这个名称来辨认表单信息。
- value=" zhangsan ":设定该文本框的默认值。若不填,则文本框是空白的,等待用户输入;若像例子中那样设定 value= " zhangsan ",则 text 便会出现在文本框中。用户可以修改。
- align = "middle ":设定文本框放置的位置。可选值为:top,middle,buttom,left,right,texttop,baseline,absmiddle。
- size= "10 ":设定文本框显示的长度。
- maxlength= "255 ":设定文本框允许输入英文字符数的上限。为方便编排资料或避免错误输入,宜设定上限,例如电话可设定为 11 位(包括区号),年龄可设为 2 等。

(2) 密码框(password)。

例如:<input type = "password" name = "password" value = " 123456" align = "middle" size= "10" maxlength= "20 ">

- type= "password":输入方式为 password,能产生一个密码框,上限为 255 个英文字符。
- name、value、align、size、maxlength 属性的设置与 text 的属性设置相同。

(3) 隐藏域(hidden)。

例如:<input type= "hidden" name= "hidden" value="hidden">

- type="hidden":输入方式为隐藏域,它不会在页面上显示任何信息,但其值会随表单一起传给处理表单的应用程序。
- name = "hidden ":设定隐藏域的名称,供应用程序识别。
- value= "hidden":设定隐藏域的值。该值在提交表单时会与表单数据一起传给应用程序。

(4) 文件域(file)。

例如:<input type= "file" name= "file" align=" buttom" size = "20" maxlength =" 100" accept= "text/html ">

- type= "file":输入方式为 file,产生一个文件域。
- name= "file":文件域的名称,供应用程序识别。
- align= "bottom":设定文件域摆放的位置。可选值为:top,middle,buttom,left,right,texttop,baseline,absmiddle。
- size= "20":设定文本框的显示长度。
- maxlength = "100 ":设定文本框可以输入字符数的上限。
- accept= "text/html ":设定文件域所接受的文件类别。

(5) 单选按钮(radio)。

例如:<input type = "radio" name = "radio" value = "radio" align = "middle" checked>。

- type= "radio":输入方式为 radio,能产生一个单选按钮。
- value= "radio":设定单选按钮的默认值。每个单选按钮必须有且仅有一个 value,

通常有同时采用两个或两个以上同 name,而不同 value 的 radio 输入方式,让用户任选其一,这样它们就组成了一个单选按钮组。

- name 、align 属性的设置与 text 的属性设置相同。
- checked:设定该单选按钮为默认被选中。同 name 的各个单选按钮中只能有一个使用或全部不使用这个属性参数。

(6) 复选框(checkbox)。

例如:<input type="checkbox" name=" checkbox" value=" checkbox" align="left" checked>。

- type= "checkbox ":输入方式为 checkbox,能产生一个复选框。
- value= "checkbox ":每个复选框必须有且仅有一个 value,当选中时,这个值便会传递给处理表数据的应用程序。
- name、align 属性的设置与 text 的属性设置相同。
- checked:设定该复选框为默认被选中。同 name 的各个复选框都可以使用这个属性参数。

(7) 提交按钮(submit)和清除按钮(reset)。

例如:<input type= "submit" value="提交" align="middle">。

<input type= "reset" value="清除" align= "middle">。

- type=" submit"或 type = "reset":输入方式为 submit 或 reset,能产生一个提交按钮或清除按钮。
- value="确定"或 value="清除":这个值不是提交给应用程序的,而是显示在按键上。
- align= "middle":设定按钮放置的位置。

(8) 普通按钮(button)。

例如:<input type= "button" value= "button">。

- type="button":输入方式为普通按钮,通常需要 Javascript 来辅助提交表单中的数据。
- value 属性的设置与 submit 的属性设置相同。

6. 其他常用标签

除了以上介绍的标签外,还有一些常用的标签。表 2-5 是常用的 HTML 标签。

表 2-5 HTML 常用标签表

开始标签	结束标签	含 义
<img>	无	图像标签
<a>		超链接标签
<meta>	</meta>	Web 页的其他信息
<style>	</style>	样式标签
<link/>	无	链接到外部文件
<frameset>	</frameset>	一组框架
<frame>	</frame>	一组框架中的一个框架

其中,各标签的具体含义如下:

(1) <img>图像标签,常用属性参数

src 设定图片的来源,接受.gif,.jpg,.png 格式。如果图片与该 HTML 文件处在同一目录,则只填写文件名称,否则必须加上正确的路径,相对路径或绝对路径皆可。

width 设定图片的宽度,一般采用像素作为单位。

height 设定图片的高度,一般采用像素作为单位。

border 设定图片边框的厚度。

align 旁边文字的位置(可选值为:top,middle,bottom(默认值),left,right)。

alt 若是使用文字浏览器,由于不支持图片,这些文字会代替图片被显示。若浏览器支持图片显示,当鼠标移至图片上这些文字也会显示。

(2) 超链接标签<a>,常用属性参数

href 设定该链接所要连到的资源文件名称,若该文件与此 HTML 文件不是同一目录,加上适当的路径,相对绝对皆可。

target 设定被按下后结果所要显示的视窗(可选值为:_blank,_parent,_self(默认值),_ top,framename),即在新窗口中,打开被链接文档,在父框架中打开被链接文档,在被点击时的同一框架中打开被链接文档,在窗口主体中打开被链接文档,在指定页框链接文档。

(3) 页面信息标签<meta>,常用属性参数

name 指定待设定的属性名称;

content 设定文档的内容类型,HTML 是"text/html";

http- equiv 指定待设定的 HTTP 报头名称。

(4) 样式标签<style>用于指定样式文件的位置。

(5) 连接标签<link>用来将当前文件与其他 URL 进行链接,但不会有链接按钮,用在<head>标签间。

(6) 框架组标签<frameset>表示一个框架组,常用属性参数

cols 用框架垂直分割页面(即将页面左右分开);

rows 用框架水平分割页面(即将页面上下分开);

frameborder 设定框架的边框,其值只有 0 和 1,0 表示不要边框,1 表示要显示边框;

border 设定框架的边框厚度,以像素为单位;

bordercolor 设定框架的边框颜色;

framespacing 设定框架之间保留的空白距离。

(7) 框架标签<frame>,表示框架组中的单个框架。

src 设定在框架中显示的文档的位置。如果显示的文档与该 HTML 文件处在同一目录,则只填写文件名称,否则必须加上正确的路径,相对路径或绝对路径皆可。

示例2-5

```
/**
 * HTML 框架标签演示代码
 */
```

```
<html>
<frameset rows="40%,60%" cols="*">
    <frame src="top.html">
    <frameset cols="25%,75%">
        <frame src="left.html">
        <frame name="right_frame" src="right.html">
    </frameset>
</frameset>
</html>
```

```
/**
 * top.html
 */
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>img</title>
</head>
<body>
<img src="top.jpg" width="800" height="200" alt="温暖的雪夜" />
</body>
</html>
```

```
/**
 * left.html
 */
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>img</title>
</head>
<body>
<a href="right.jpg" target="right_frame">在右面窗口显示图片</a>
</body>
</html>
```

```
/**
 * right.html
 */
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>img</title>
</head>
<body>
我是右面窗口,嘻嘻嘻
```

```
</body>
</html>
```

示例 2-5 的效果分别如图 2-8、图 2-9 所示。



图 2-8　示例 2-5 的效果图



图 2-9　单击“在右面窗口显示图片”后效果图

2.2 CSS 样式表

CSS(Cascading Style Sheets)是级联样式表或层叠样式表的简称，用来控制页面元素的字体、字号、色彩、背景、间距以及大小、位置等格式属性，从而制作出丰富多彩的网页效果。使用 CSS 样式表控制页面的显示效果，可使网站的页面维持统一风格。

DIV＋CSS 页面布局模式是 W3C 标准的一个典型应用，也是目前主流页面的布局模式。



说 明

一般的页面布局方式：表格布局(使用表格实现页面的整体布局)、DIV＋CSS 页面布局。

使用嵌套的表格布局，HTML 层次结构复杂，代码量非常大，但是表格布局结构具有相对稳定、简单通用的优点，所以表格数据一般适用于页面中数据规整的局部布局，而页面整体布局一般采用 DIV＋CSS 页面布局模式。

2.2.1 在网页中使用 CSS 的三种方式

按照 CSS 代码在网页文件中出现的不同位置，可以将 CSS 划分为行内样式、嵌入样式和外部样式 3 种方式。

1. 行内样式

又称为内嵌样式，通过设定 HTML 标签的 style 属性值，嵌入 CSS 代码。例如：

```
<p style="color:red;">hello</p>
```

2. 嵌入样式

在页面的头部使用<style> </style>标签嵌入 CSS 代码

3. 外部样式

将 CSS 代码编写在一个独立的文件中，该文件的后缀名是.css，在页面中使用<link>标签或<style>…</style>标签来指明所需使用的样式文件。例如 CSS 文件名为 mySytle.css，那么在页面中引入该样式的代码为

```
<link rel="stylesheet" href="myStyle.css" style="text/css"/>
```

或者

```
<style type="text/css">
        @import url(myStyle.css);
</style>
```

这种形式使用最为普遍，便于多个页面保持统一风格。

2.2.2 CSS 选择器

CSS 通过为指定的元素定义样式来达到样式控制的目的，定义 CSS 样式的格式为

```
选择器{
    属性名称 1: 属性值 1;
    属性名称 2: 属性值 2;
    …
}
```

选择器指明样式影响到页面上哪些 HTML 元素，基本的选择器有 4 种：

1. 标签选择器

通过 HTML 的标签名称来指定样式。例如，CSS 代码 P{color:red;}表示将所有的<p>标签中的文字设置为红色。

2. id 选择器

若为页面的一个元素指定了 id，则可以通过 id 单独为其定义 CSS 样式，一般来说，元素的 id 在整个页面中应该是唯一的，通过 id 定义的 CSS 的语法格式如下：

```
#id 名称{
    属性名称 1: 属性值 1;
    属性名称 2: 属性值 2;
    …
}
```

注意 id 名称前应加上前缀“#”.

3. 类选择器

若页面上的多个元素要使用相同的样式，则可以通过 class 属性为这些元素指定相同的类名，然后为该类定义一组样式。

```
.class 选择器名称{
    属性名称 1: 属性值 1;
    属性名称 2: 属性值 2;
    …
}
```

注意类选择器名称前应加上前缀“.”。

4. 伪类选择器

伪类是特殊的类，能自动地被支持 CSS 的浏览器所识别。伪类可区别标记的不同状态。伪类不用 HTML 的 class 属性来指定。伪类的定义格式如下：

```
选择器: 伪类{
    属性名称 1: 属性值 1;
    }
```

伪类的一个最常见的应用是指定超链接<a>以不同的方式显示链接(links)、悬浮链接(hover)、已访问链接(visited)和可激活链接(active)。

示例2-6

```
/**
 *CSS 样式演示代码
 */
<html>
```

```
<head>
<title>CSS 样式</title>
<style>
    a{
    font-style: italic;
    font-size: 24px;
    }
    a:hover {
        font-size: 26px;
        font-style: normal;
        text-decoration: underline;
    }
    p{
        font-size: 12px;
    }
    #today{
    color:blue;
    background-color:#FFCCFF;
    font-family:隶书;
    font-size:24px;
    }
    .now{
font-size: 28px;
}
</style>
</head>
<body>
    <p style="color:red; font-size:16px; ">你所浪费的今天,</p>
    <div id="today">是昨天死去的人奢望的明天; </div>
    <p class="now">你所厌恶的现在,</p>
    <p><a href="#">是未来的你回不去的曾经.</a></p>
</body>
```

示例 2-6 的显示效果如图 2-10 所示。

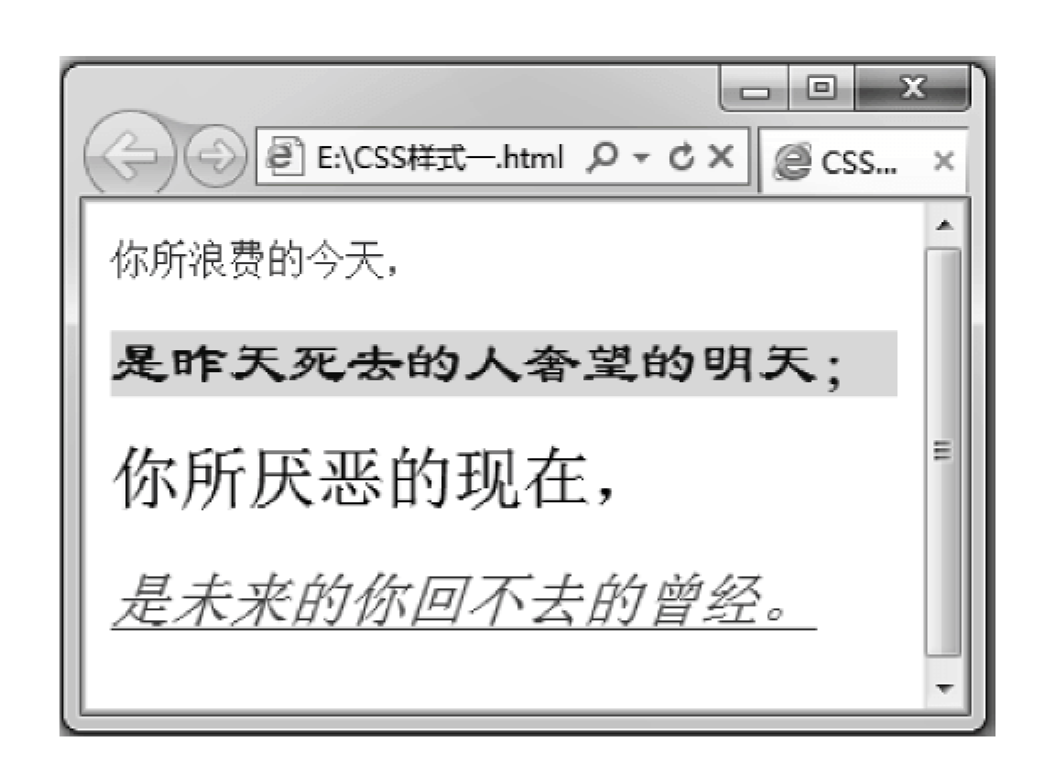


图 2-10 示例 2-6 的效果图

示例 2-6 中使用了样式的两种方式“<p style="color:red; font-size:16px; ">所浪费的

今天，</p>"为行内样式；"<style>a{…}…</style>"为嵌入样式。

在此示例中 4 种选择器全部用到，我们来分析一下，"<style>a{…}p{…}…</style>"为<a>、<p>的标签选择器；"<style># today{…} … </style>"为 id 选择器，该样式应用在"<div id="today">是昨天死去的人奢望的明天；</div>"；"<style> . now { …} … </style>"为类选择器，样式应用在"<p class="now">你所厌恶的现在，</p>"；"<style> a：hover { …}…</style>"为伪类选择器，样式应用在"<p>是未来的你回不去的曾经。<p>"；当鼠标悬浮在链接上时，效果如图 2-11 所示。

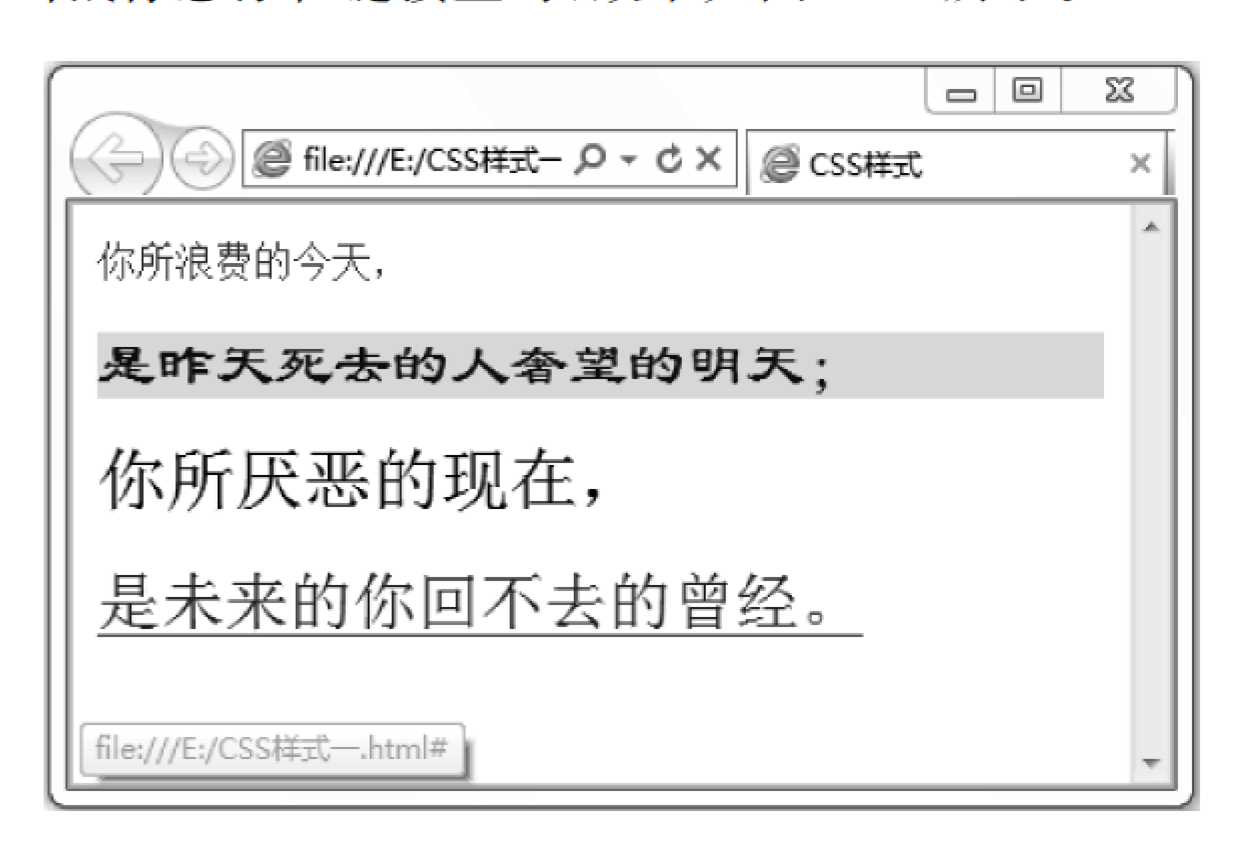


图 2-11 示例 2-6 的 a:hover 效果图

需要说明的是，如果是标签嵌套，则应用最里层的样式。由示例 2-6 看到，样式嵌套在文件中，使得文件条理不清晰，因此我们可以采用外部样式表，如示例 2-7。

示例2-7

```
/* *
 * HTML 应用外部样式表演示代码
 * /
<html>
<head>
<title>CSS 样式</title>
<link rel = "stylesheet" href = "myStyle.css" style = "text/css"/>
</head>
<body>
    <p class = "first">你所浪费的今天,</p>
    <div id = "today">是昨天死去的人奢望的明天; </div>
    <p class = "now">你所厌恶的现在,</p>
    <p><a href = "#">是未来的你回不去的曾经.</a><p>
</body>
</html>
```

```
/* *
 * CSS 样式文件 myStyle.css
 * /
    a{
```

```
        font - style: italic;
        font - size: 24px;
    }
    a:hover {
        font - size: 26px;
        font - style: normal;
        text - decoration: underline;
    }
    p{
        font - size: 12px;
    }
    #today{
        color:blue;
        background - color: #FFCCFF;
        font - family:隶书;
        font - size:24px;
    }
    .now{
        font - size: 28px;
    }
    .first{
        color:red;
        font - size:16px;
    }
```

效果与图 2-10、图 2-11 一致。效果相同,可以使 HTML 页面变得简洁、清晰。若多个网页采用同一个 CSS 样式文件,则这些网页的风格将保持一致。

2.3 JavaScript 简介

我们为什么还要学习 JavaScript 呢? 原因是：JavaScript 可以实现非常多的功能,例如,客户端表单验证、页面动态效果、动态改变页面内容等。那么 JavaScript 是什么? JavaScript(简称 JS)既是一种描述语言,也是一种基于对象(Object)和事件驱动,并具有安全性能的客户端脚本语言,使用它的目的是与 HTML 超文本标记语言一起实现在一个 Web 页面中链接多个对象,与 Web 客户实现交互。

2.3.1 脚本的基本结构

通常,JavaScript 代码是用<script>标记嵌入 HTML 文档中。可以将多个脚本嵌入到一个文档中,只需将每个脚本都封装在<script>标记中即可。浏览器在遇到<script>标签时,将逐行读取内容,直到遇到</script>结束标记为止。然后浏览器将检查 JavaScript 语句的语法,如果有任何错误,就会在警告框中显示;如果没有错误,浏览器将编译执行语句。

脚本的基本结构如下：

```
<script language = "JavaScript" type = "text/JavaScript">
        JavaScript 语句;
```

```
</ script >
```



说 明

根据 W3C，HTML4.01 规范和 XHTML1.0 规范。script 标记可以带有的属性中：

type 属性是必须的(Required)，而 language 属性是可选的(Implied)，因此虽然两者表面看上去用谁都无所谓，但是具有 type 属性是符合 Web 标准的。

type 是在 html4.0 增加的，而 language 就是比较早以前的了。建议使用< script type ="text/Javascript">

2.3.2 脚本的执行原理

了解了脚本的基本结构，下面再来深入了解脚本的执行原理。

在脚本的执行过程中，浏览器客户端与应用服务器端采用请求/响应模式进行交互，如图 2-12 所示。

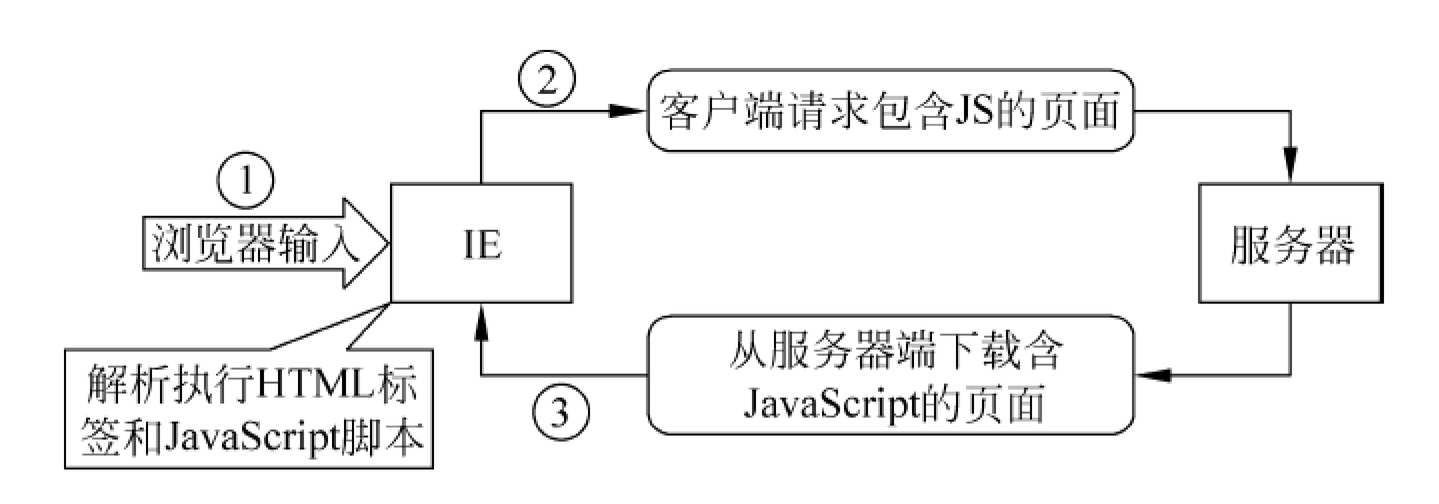


图 2-12 脚本执行原理

现在，让我们逐步分解一下这个过程。

1. 浏览器接收用户的请求：用户在浏览器的地址栏中输入要访问的页面(页面中包含 JavaScript 脚本程序)。

2. 向服务器端请求某个包含 JavaScript 脚本的页面，浏览器把请求消息(要打开的页面信息)发送到应用服务器端，等待服务器端的响应。

3. 应用服务器端向浏览器发送响应消息，即把含有脚本的 HTML 文件发送到浏览器客户端，然后由浏览器从上至下逐条解析 HTML 标签和 JavaScript 脚本，并将页面效果呈现给用户。

使用客户端脚本的好处，首先含脚本的页面只要下载一次即可，这样能减少不必要的网络通信。其次脚本程序是由浏览器客户端执行，而不是由服务器端执行，因此能减轻服务器端的压力。

下面通过一个简单的例子来讲解脚本的基本结构和执行原理。

示例2-8

```
/**
 *JavaScript 代码
 */
```

```
<html>
<head>
<title>JavaScript 应用</title>
<script type="text/JavaScript">
    document.write("使用 JavaScript 脚本循环输出 Hello JSP");
    for(var i=0;i<5;i++){
        document.write("<h3>Hello JSP</h3>")
    }
</script>
</head>
<body>
</body>
</html>
```

示例 2-8 的效果如图 2-13 所示。



图 2-13　示例 2-8 的效果图

示例 2-8 中“document. write()”是标准的 JavaScript 命令，用来向页面输出内容。把 document. write() 命令输入到<script>与</script>之后。浏览器就会把它当做一条 JavaSript 命令来执行，这样浏览器就会向页面写入“Hello JSP”。

我们已经通过简单的例子了解了脚本的基本结构和脚本的执行原理，并且已经知道可以通过使用<script>…</script>的方式在网页中使用 JavaScript，那么在网页中还有其他的方式可以引用 JavaScript 吗？答案是肯定的。JavaScript 作为客户端程序，嵌入网页有以下 3 种方式。

1. 使用<script>标签，示例 2-8 即为使用<script>标签；

2. 使用外部 JavaScript 文件，使用外部 JavaScript 文件是将 JavaScript 写入一个外部文件中，以 *.js 为后缀保存这个文件。

我们来看下面代码：

```
<html>
<head>
<script src="hello.js" type="text/JavaScript"></script>
```

```
</head>
<body>
</body>
</html>
```

3. 直接嵌入在 HTML 标签中。

直接在 HTML 标签中嵌入代码如下：

```
<input name="btn" type="button" value="弹出消息框" onclick="javascript:alert
('欢迎你')";/>
```

当单击“弹出消息框”按钮时，弹出对话框，如图 2-14 所示。



图 2-14 提示对话框

2.3.3 JavaScript 的组成

JavaScript 由三部分组成：核心（ECMAScript）、文档对象模型（Document Object Model，简称 DOM）、浏览器对象模型（Browser Object Model，简称 BOM），如图 2-15 所示。

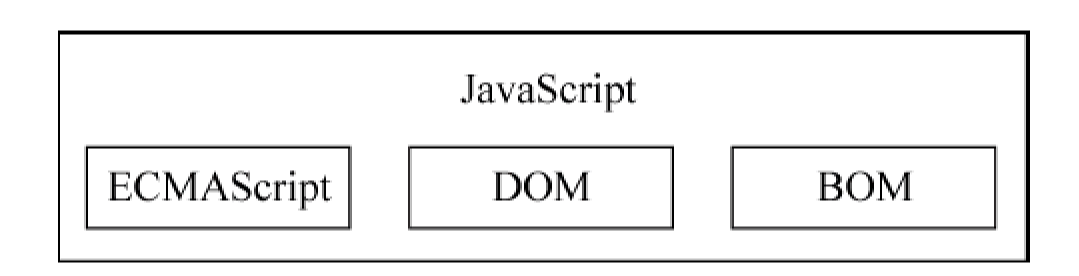


图 2-15 JavaScript 的组成

1. ECMAScript 为 JavaScript 的核心，描述了语言的基本语法和对象。

ECMAScript 主要提供语言相关的信息与标准，如语法、类型、声明、关键字、保留字、操作运算符、对象等。

2. DOM，当网页被加载时，浏览器会创建页面的文档对象模型 DOM。HTML DOM 模型被构造为对象树，见图 2-16。

3. BOM，提供了独立于内容而与浏览器窗口进行交互的对象。由于 BOM 主要用于管理窗口与窗口之间的通信，因此其核心对象是 window。

BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

如图 2-17 所示，window 对象是 BOM 的顶层（核心）对象，所有对象都是通过它延伸出来的，也可以称为 window 的子对象。由于 window 是顶层对象，因此调用它的子对象时可以不显示的指明 window 对象，例如下面两行代码是一样的：

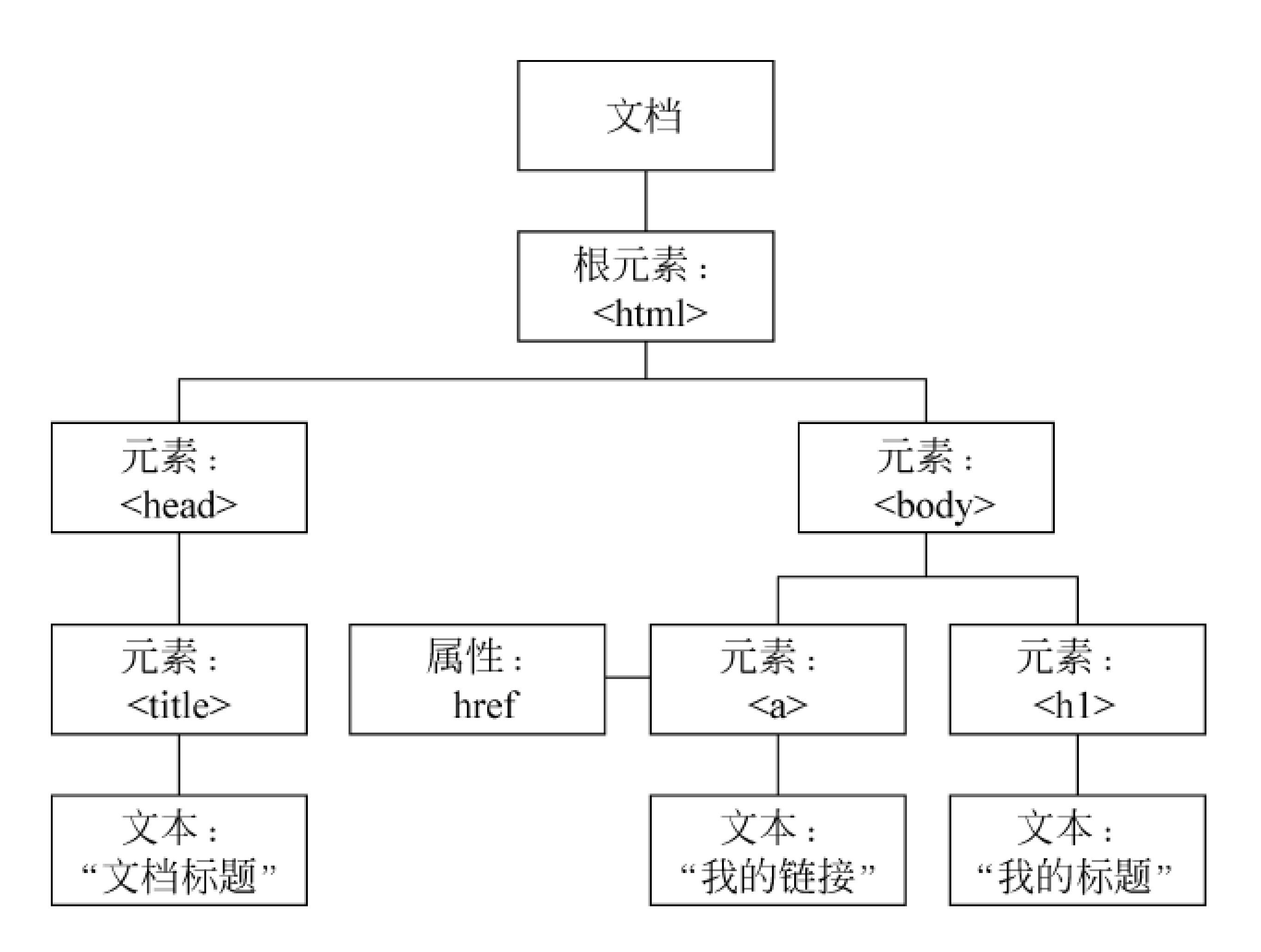


图 2-16 DOM 对象树

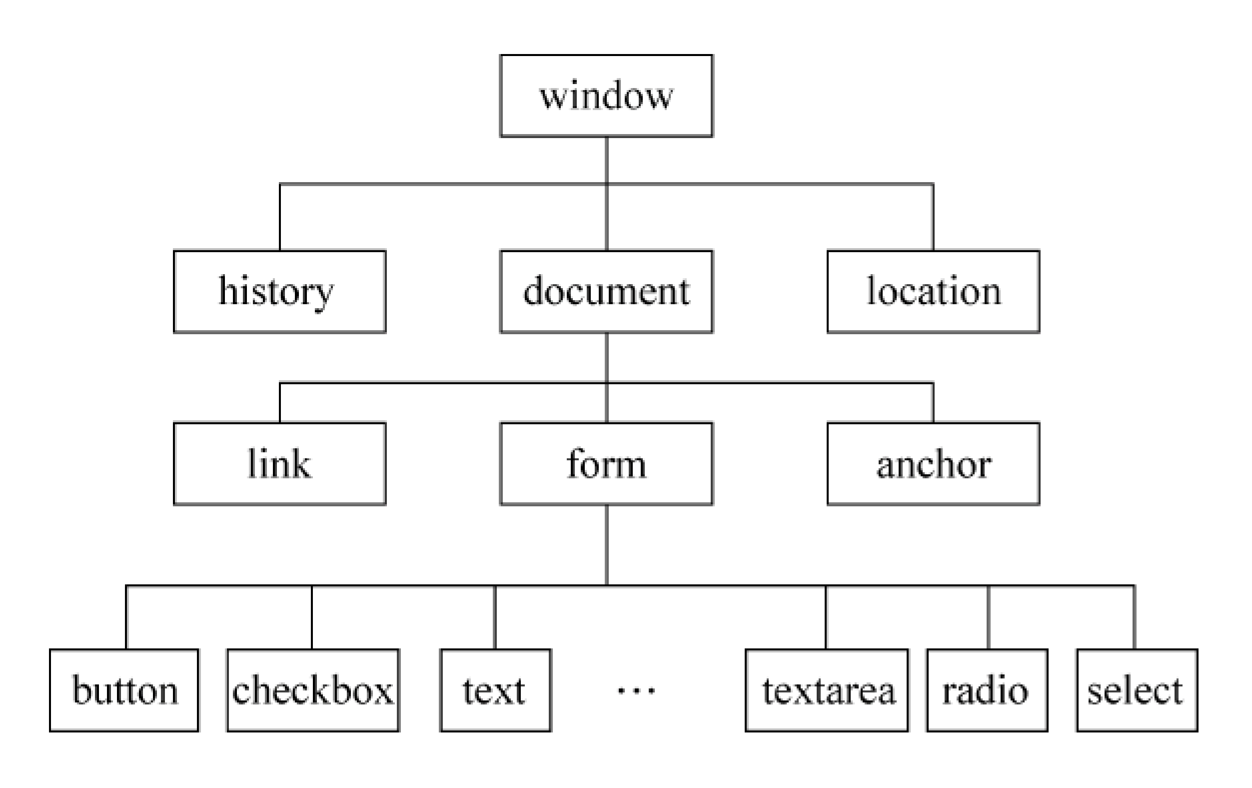


图 2-17 BOM 结构图

```
document.write("hello jsp");
window.document.write("hello jsp");
```

2.3.4 JavaScript 核心语法

JavaScrip 也是一门编程语言，也包含变量的声明、赋值、符号运算、逻辑控制语句等基本语法，这里只是简单的介绍，在以后的例子中结合程序再具体解释其作用。

1. 变量声明与赋值

JavaScript 是一种弱类型语言，没有明确的数据类型，也就是说，在声明变量时，无需指定变量的类型，变量的类型由赋值变量的值决定，下面是 JavaScript 声明变量的语法格式

```
var 合法变量名;
```

其中 var 是声明变量的关键字，JavaScript 的命名规则和 Java 变量的命名规则相同。例如：

```
var width = 20;//声明变量 width 的同时，将数值 20 赋给变量
```

需要强调的是，JavaScript 区分大小写，所以大小写不同的变量名表示不同的变量。另外，由于 JavaScript 是弱类型语言，所以允许不声明变量而直接使用，系统将会自动声明该变量，例如：

```
X = 88;//没声明变量 x,而直接使用.
```

经验总结

- 千万要注意 JavaScript 区分大小写，特别是变量的命名、语句关键字等，这种错误有时很难查找。
- 变量可以不经声明而直接使用，但是这种方法很容易出错、也很难查找排查错误、不推荐使用。在使用变量前，先声明后使用，这时良好的编程习惯。

2. 数据类型与运算符

虽然 JavaScrip 是一种弱类型语言，但是在 JavaScript 中也提供了基本的数据类型，如：undefined（未定义类型），null（空类型），number（数值类型），string（字符串类型），boolear（布尔类型）。

JavaScrip 中常见的运算符可分为算术运算符、比较运算符、逻辑运算符和赋值运算符，如表 2-6 所示。

表 2-6　常用运算符表

类　　型	运　算　符
算术运算符	+　-　*　/　%　++　--
赋值运算符	=
比较运算符	>　<　>=　<=　==　! =
逻辑运算符	&&　||　!

3. 逻辑控制语句

在 JavaScrip 中，逻辑控制语句用于控制程序的执行顺序，同 Java 一样，逻辑控制语句也分为两类：条件结构和循环结构。

(1) 条件结构

① if 结构

基本语法为

```
If(表达式){
    JavaSript 语句 1;
}
else{
    JavaSript 语句 2;
}
```

② swith 结构

基本语法为

```
swith(表达式){
    case 值 1:
        JavaSript 语句 1;
        break;
    case 值 2:
        JavaSript 语句 2;
        break;
        ….
    default:
        JavaSript 语句 n;
        break;
}
```

(2) 循环结构

同 Java 一样,JavaScript 中的循环结构也分为 for 循环、while 循环、do-while 循环。

① for 循环

基本语法如下:

```
for(初始化;条件;增量或减量){
JavaScript 语句;
}
```

② while 循环语句

基本语法如下:

```
while(条件){
    JavaScript 语句;
}
```

③ do-while 循环语句

```
do{
    JavaScript 语句;
}while(条件);
```

④ 中断循环

```
break,continue
```

4. 注释

注释是描述部分程序或整个程序功能的一段说明性文字,注释不会被解释器执行,而是直接跳过。注释的功能是帮助开发人员阅读、理解、维护和调试程序。JavaScript 语言的注释与 Java 语言的注释一样,分为单行注释和多行注释两种。

单号注释以//开始,以行末结束。例如

```
alert("用户名密码不匹配");//在页面中弹出登录验证信息
```

其中 alert 是警告,alert("提示信息")方法会创建一个特殊的小窗口,该窗口带有一个字符串和一个确定按钮。

多行注释以/ * 开始,以 * /结束。

5. 函数

在 JavaScript 中,函数类似于 Java 中的方法,是执行特定任务的语句块,但是 JavaScript 中的函数使用更简单,不用定义函数属于哪个类,可以直接调用函数名来使用函数。JavaScript 中提供了一些常用的函数,例如:parseInt()可以解析一个字符串,并返回一个整数,isNaN()是用于检验其属性参数是否为非数字等。

下面我们重点来学习自定义函数和调用函数。

(1) 创建函数

语法格式如下:

```
function 函数名(属性参数 1,属性参数 2,属性参数 3….){
    JavaScript 语句;
}
```

属性参数 1,属性参数 2 等是传入函数的变量或值,“{”,“}”定义了函数的开始和结束。

函数中的属性参数是可选的,当传入属性参数时通常把函数称为有参函数,当没有传入属性参数时通常把函数称为无参函数,无参函数必须在其函数名后加括号,例如

```
function 函数名(){
    JavaScript 语句;
}
```

在 JavaScript 中 return 语句用来规定从函数返回的值,因此需要返回某个值的函数必须使用 return 语句。

(2) 调用函数

要执行一个函数,必须先调用这个函数,当调用函数肘,必须指定函数名及其后面参数(如果有属性参数)。函数的调用一般和表单元素的事件结合使用,调用格式如下:

```
事件名 = "函数名()";
```

下面通过示例 2-9 来学习如何创建函数和调用函数。

示例2-9

```
/* *
 * JavaScript 函数
 * /
<html>
<head>
<title>无参函数—使用函数显示 Hello JSP 5 次</title>
<script type = "text/javascript">
function showHello(){
    for(var i = 0;i<5;i++){
    document.write('<h2>Hello JSP</h2>')
```

```
    };
  }
</script>
</head>
<body>
<input name="btn" type="button" value="显示 5 次 Hello JSP" onclick="showHello()"/>
</body>
```

showHello()是创建的无参函数。onclick 表示按钮单击事件,当单击按钮时调用函数 showHello()。

显示效果如图 2-18、图 2-19 所示。

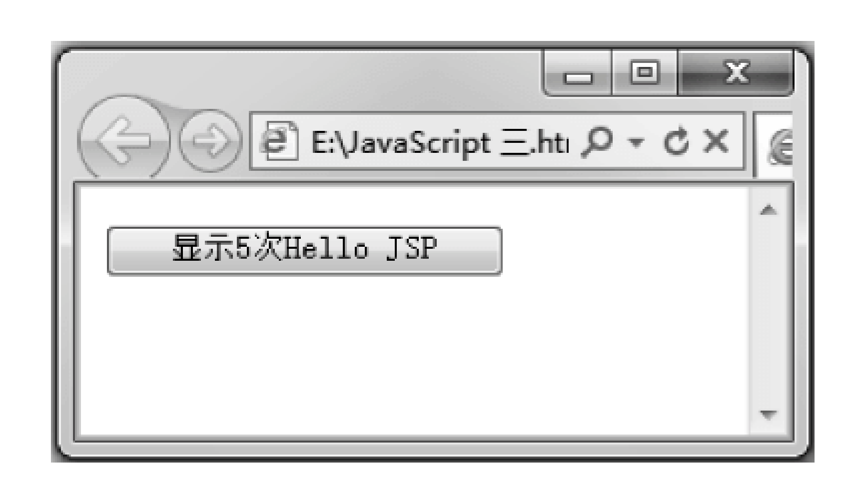


图 2-18 示例 2-9 效果图



图 2-19 单击"显示 5 次 Hello JSP"按钮后

6. 对象

JavaScript 的一个重要功能就是面向对象的功能,通过基于对象的程序设计,可以用更直观、模块化和可重复使用的方式进行程序开发。一组包含数据的属性和对属性中包含数据进行操作的方法,称为对象,比如要设定网页的背景颜色,所针对的对象就是 document,所用的属性名是 bgcolor,如 document.bgcolor="blue",就是表示使背景的颜色为蓝色。

7. 事件

用户与网页交互时产生的操作,称为事件。事件可以由用户引发,也可能是页面发生改变,甚至还由看不见的事件引发。绝大部分事件都由用户的动作所引发,例如:按鼠标的按键就产生 onclick 事件,示例 2-9 即使用了 onclick 事件。

2.3.5 JavaScript 表单验证

示例2-10

```
/**
 * JavaScript 实现简单表单验证
 */
<html>
<head>
```

```
<title>JavaScript 简单表单验证</title>
<script type="text/javascript">
function check(){
    var name = document.getElementById('name').value;
    var password = document.getElementById('password').value;
    if(name == null || name == ''){
        alert('请输入用户姓名');
        return false;
    }
    if(password == null || password == ''){
        alert('请输入密码');
        return false;
    }
    else return true;
    }
</script>
</head>
<body>
<form method="post" name="myForm" action="" onSubmit="return check()">
用户名:<input type="text" id="name" name="txtName"><br/>
密码: <input type="password" id="password" name="txtPassword"><br/>
<input type="submit" value="登录"><input type="reset" value="重置">
</form>
</body>
</html>
```

效果如图 2-20 所示。

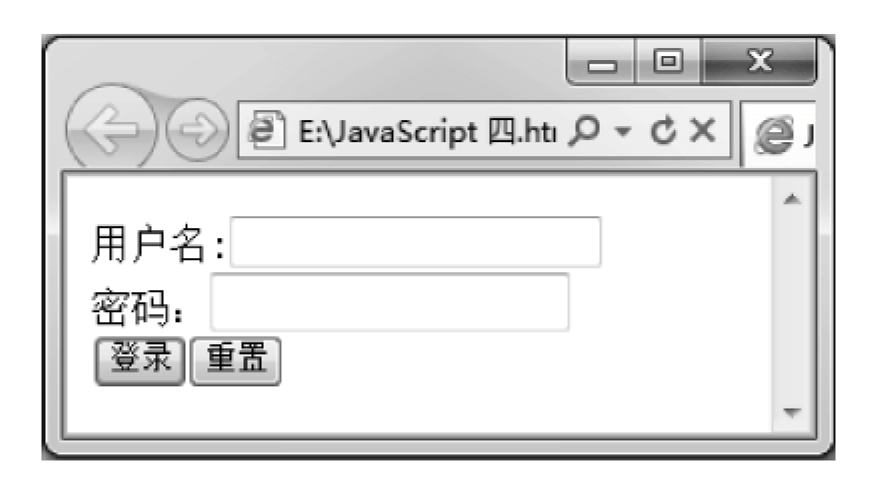


图 2-20　示例 2-10 的效果图

若用户名、密码为空时弹出的对话框如图 2-21、图 2-22 所示。







图 2-21　用户名为空时弹出的对话框　　图 2-22　密码为空时弹出的对话框

当用户在表单中单击了提交按钮后，则发生提交事件。编写 JavaScript 代码对表单的提交事件进行处理，可以在客户端实现对表单数据的验证。由于 JavaScript 代码是由浏览器解释执行的，因此用 JavaScript 实现的数据验证也被称为“客户端验证”或“前端验证”。这时表单的一般形式为：具体的验证工作则在函数 check 中进行。当 check 函数的返回值为 false 时，提交动作被阻止；若为 true 时，则将对指定的资源(由 form 标签的 action 属性指出，如 JSP 页面)发出请求，同时把表单数据和其他请求信息一起提交给该资源。这里应注意，onSubmit 属性的值为"return check ()"，而非"check()"，后面这种写法不能起到拦截提交动作的作用。

在 JavaScript 中，元素的 value 属性表示该元素的输入值，获取此值的方法有两种：若有一个元素的 id 为"name"，则获取该元素值的表达式是 document. getElementByld (" name"). value，该表达式的值是字符串类型；如示例 2-10，要获取用户名后文本框的值的表达式为 document. myForm. txtName. value。

2.4 上机练习

1. 创建如图 2-23 所示的 HTML 页面 exercise1. html。

需求说明：该页面中使用文本标签＋列表标签＋行内样式。

实现思路：参照 2. 1. 2. 1、2. 1. 2. 2、2. 2. 1 节。

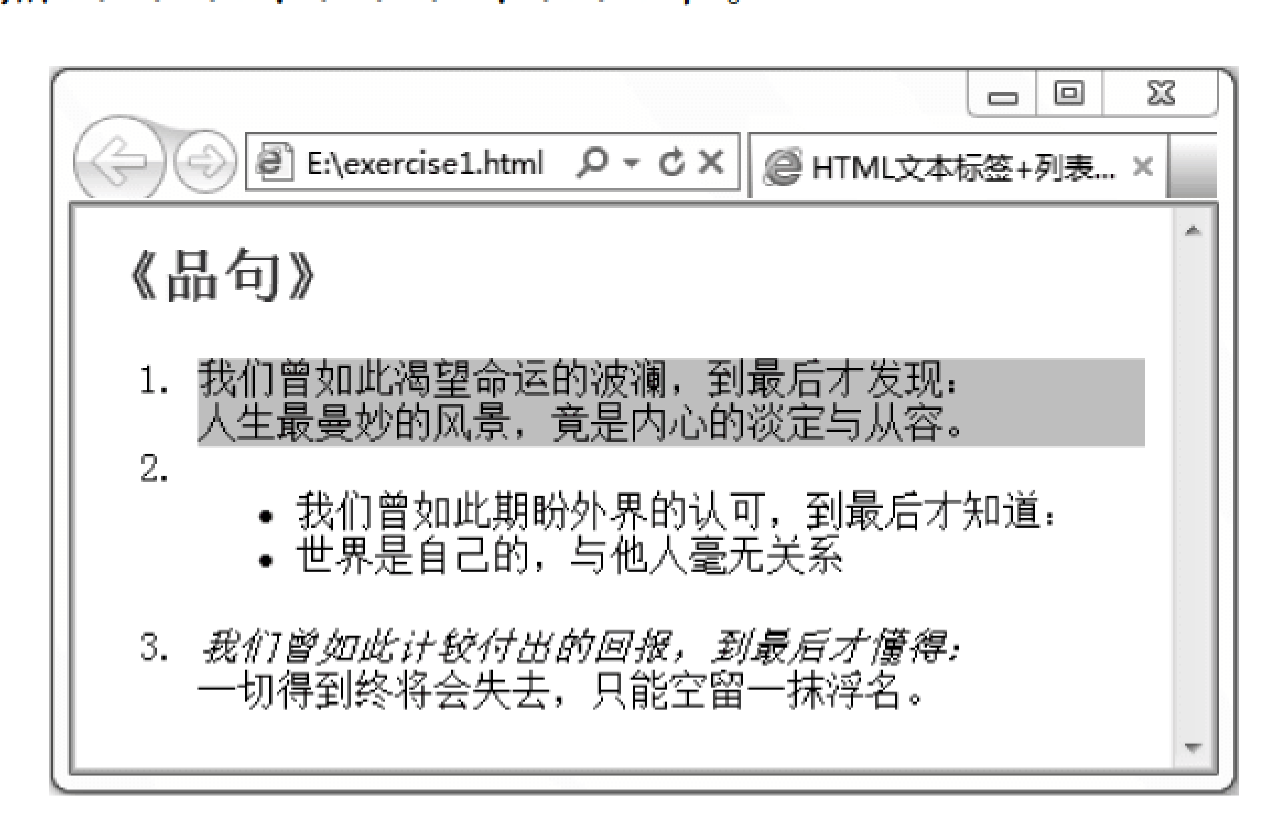


图 2-23 文本标签＋列表标签＋行内样式页面效果图

2. 创建如图 2-24 所示的 HTML 页面 exercise2. html。

需求说明：该页面中使用表格标签＋内嵌样式。

实现思路：参照 2. 1. 2. 3、2. 2. 2 节。

3. 实现通知公告发布系统后台添加信息页面，创建如图 2-25 所示的 HTML 页面 exercise3. html 及 CSS 文件 myCSS. css。

需求说明：该页面中使用表单标签＋表格标签＋外部样式。

实现思路：为了布局整齐，采用表单里嵌套表格的方式；此表中的边框为细边框，细边框的设置为

```
table,table td,table th{border:1px solid #000000; border-collapse:collapse;}
```



学生学号	学生姓名	课程	成绩
1	张三	线性代数	88
2	李四	线性代数	90
3	王五	线性代数	70
总成绩			

图 2-24 表格标签＋内嵌样式页面效果图



图 2-25 表单标签＋表格标签＋外部样式页面效果图

4. 创建如图 2-26 所示的 HTML 页面 index.html。



图 2-26 通知公告发布系统 index.html 页面效果图

需求说明：通知公告发布系统的前台页面。

实现思路：这个上机练习即通知公告发布系统的前台页面分为 4 个部分：

1. 上部设置背景颜色，上部左侧一个 div，放置图片，上部右侧放置文字；
2. 中间部分的左侧显示通知公告类别；
3. 中间部分的右侧显示通知公告列表；
4. 下部为版权部分。

整个页面使用 DIV+CSS 布局。

5. 创建如图 2-27 所示的后台登录页面 login. html。

需求说明：通知公告发布系统的后台登录页面；编写 JavaScript 代码对 login. html 中的用户名和密码进行非空验证。

实现思路：利用 JavaScript 验证用户名密码非空，参照 2. 3. 4 节。



图 2-27　通知公告发布系统后台 login. html 页面效果图

6. 创建如图 2-28 所示的通知公告发布系统后台页面 back. html。



图 2-28　通知公告发布系统后台 back. html 页面效果图

需求说明：完成通知公告发布系统后台页面。

实现思路：这个上机练习即通知公告发布系统的后台主页面分为 4 个部分：

1. 上部设置背景颜色，上部左侧一个 div，放置图片，上部右侧放置文字；
2. 中间部分的左侧显示添加通知公告、通知公告列表两个连接；
3. 中间部分的右侧添加通知公告表单；
4. 下部为版权部分。

整个页面使用 DIV＋CSS 布局，局部采用 table 布局（添加信息表单）。

2.5 总　　结

（1）HTML 标签的基本结构<html><head>…</head><body>…</body></html>。

（2）HTML 的常用标签有文本标签、列表标签、表格标签、表单标签、图片标签、链接标签等。

（3）按 CSS 代码在网页文件中出现的不同位置，可以将 CSS 划分为行内样式、嵌入样式和外部样式。

（4）CSS 中基本的选择器有四种：

- 标签选择器；
- id 选择器；
- 类选择器；
- 伪类选择器。

（5）JavaScript 嵌入网页有以下三种方式：

- 使用<script>标签；
- 使用外部 JavaScript 文件；
- 直接在 HTML 标签中。

（6）浏览器在遇到<script>标签时，将逐行读取内容，直到遇到</script>结束标记为止。然后浏览器将检查 JavaScript 语句的语法，如果有任何错误，就会在警告框中显示；如果没有错误，浏览器将编译执行语句。

2.6 作　　业

一、选择题

1. HTML 的基本结构是（　　）。

A. <html><body></body><head></head></html>

B. <html><head></head><body></body></html>

C. <html><head></head><foot></ foot></html>

D. <html><head><title></title></head></html>

2. 下列说法错误的是（　　）。

A. 密码框需要设置 Input 标签 type＝ "password"

B. 设置多选的标签是<input type＝"radio">

C. 提交方法 post 比 get 更安全

D. value 属性表示初始值，可能会随着用户的操作而改变，以提交时为准

3. 列表框的默认选择属性符合规范的正确写法为(　　)。

A. selected = "selected"　　B. selected

C. checked = "checked"　　D. selected = "true"

4. 实现背景横向平铺的效果,对应的 CSS 为(　　)。

A. div{background-image: url (images/bg. gif) ; }

B. div{background: url (images/bg. gif) repeat-x;}

C. div{background: url (images/bg. gif) repeat-y;}

D. div{background: url (images/bg. gif) no-repeat;}

5. 在 Javascript 中,运行下面的代码后的返回值是(　　)。

```
var flag = true;
document.write(typeof(flag));
```

A. undefined　　B. null　　C. number　　D. boolean

二、简答题

1. 列举常见的 HTML 标签及其作用。
2. 什么是 JavaScript?
3. 什么是 CSS?
4. 选择器的主要作用是什么? 都有哪些选择器?

三、程序题

1. 编写 HTML 代码,利用表格实现表单布局,结果如图 2-29 所示。



图 2-29　登录邮箱

2. 使用 JavaScript 脚本输出如图 2-30 所示的页面。

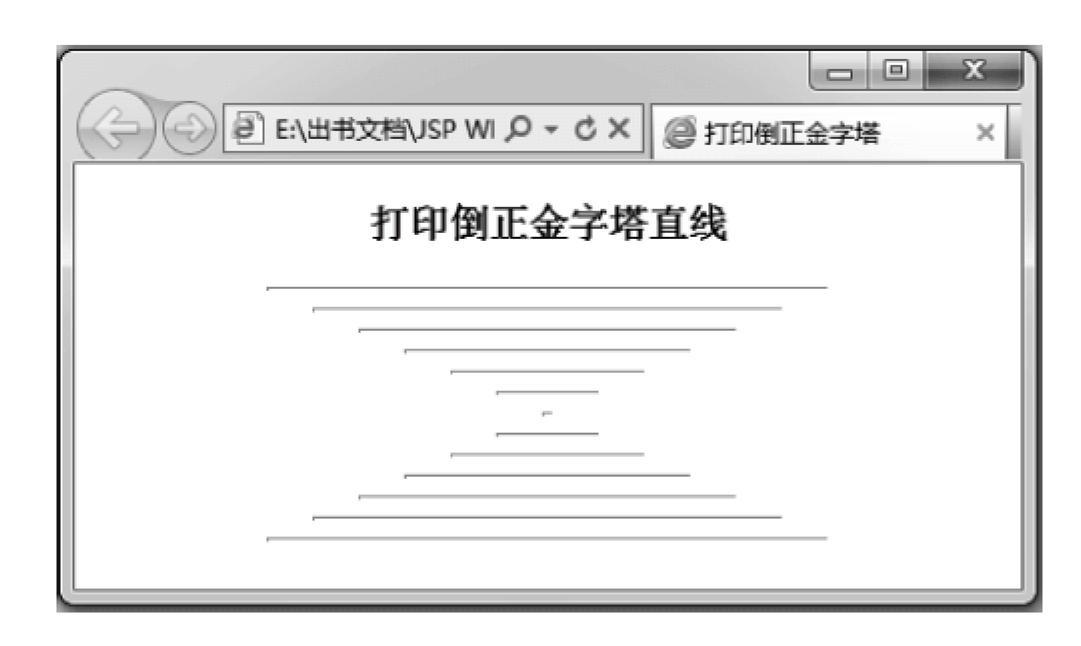


图 2-30　打印金字塔

第 3 章　JSP 基础

本章学习目标

- 理解 JSP 的工作原理
- 掌握 JSP 的三种注释方式
- 掌握 JSP 的常用指令元素：page、include、taglib
- 掌握 JSP 的脚本元素：小脚本、表达式、声明
- 掌握 JSP 的动作元素<jsp:include>、<jsp:forward>、<jsp:param>
- 掌握 include 指令和<jsp:include>的区别
- 了解 JSP 的动作元素：<jsp:plugin>

3.1　JSP 工作原理

JSP (Java Server Pages)是一种动态网页技术标准，通俗地说，JSP 就是指在 HTML 中嵌入 Java 脚本语言，当用户通过浏览器请求访问 Web 应用时，Web 服务器会使用 JSP 引擎对请求的 JSP 进行编译和执行，然后将生成的页面返回给客户端浏览器进行显示，JSP 的工作原理如图 3-1 所示。

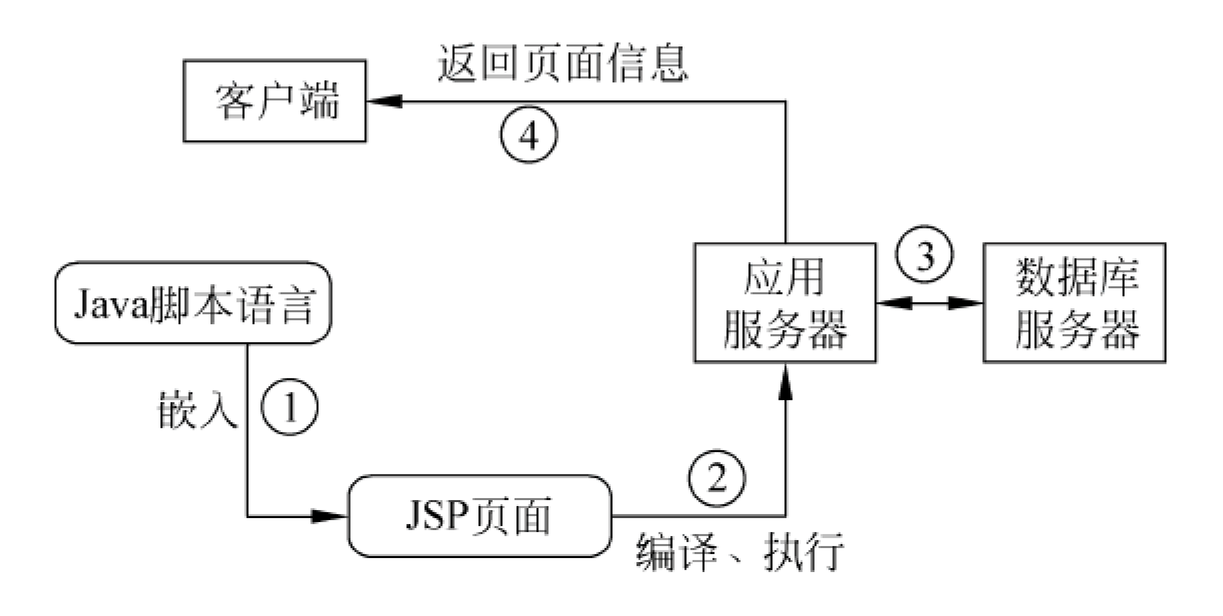


图 3-1　JSP 的工作原理图

由图 3-1 可以清晰地看到整个 JSP 的工作处理流程，那么当 JSP 提交到服务器后，服务器能够直接读出 JSP 的内容然后做出反应吗？实际上 Web 容器通过三个阶段进行处理(见图 3-2)：



资料

Web 容器是一种服务程序，在服务器一个端口就有一个提供相应服务的程序，而这个程

序就是处理从客户端发出的请求,如 Java 中的 Tomcat 容器。一个服务器可以有多个容器。

1. 翻译阶段:当 Web 服务器接收到 JSP 请求时,首先会对 JSP 文件进行翻译,将编写好的 JSP 文件通过 JSP 引擎转换成可识别的 Java 源代码,即扩展名为.java 的文件。

2. 编译阶段:经过翻译后的 JSP 文件相当于我们编写好的 Java 源文件,此时仅有源文件是不够的,还需要将 Java 源文件编译成可执行的字节码文件,即.class 文件。所以 Web 容器处理 JSP 请求的第二阶段就是编译。

3. 执行阶段:编译阶段生成字节码文件后,进入执行阶段。当执行结束后,会得到处理请求的结果。Web 容器又会再把生成的结果页面返回到客户端用户面前显示。

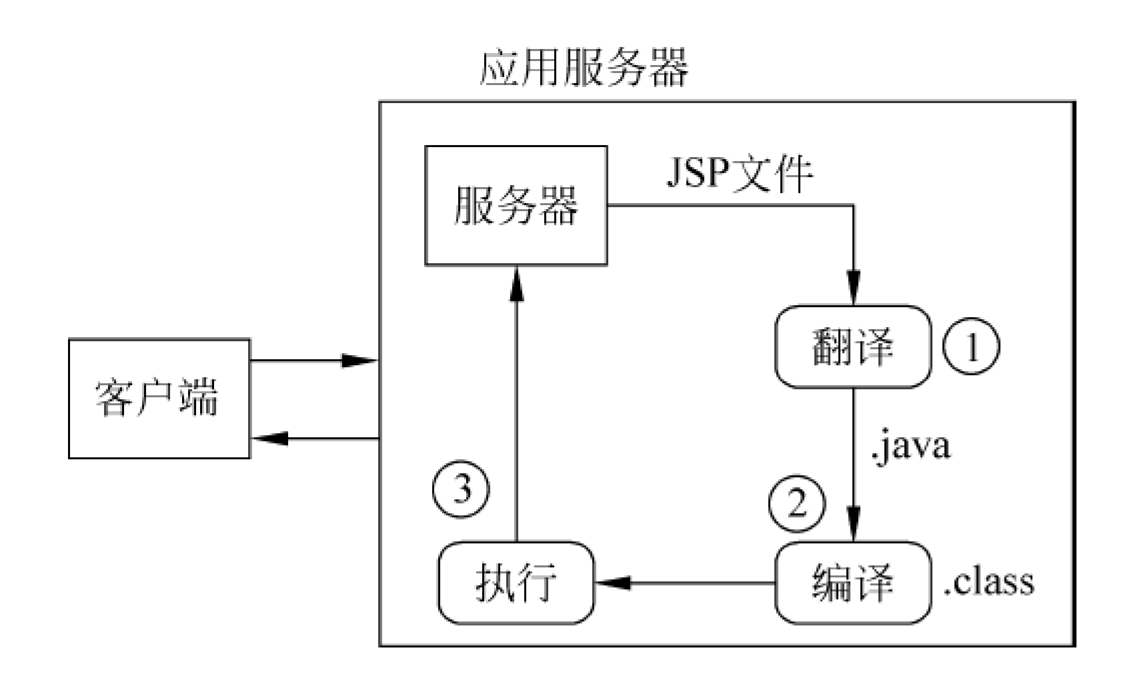


图 3-2　Web 容器处理 JSP 文件请求的三个阶段

一旦 Web 容器把 JSP 文件翻译和编译完,Web 容器会将编译好的字节码文件保存在内存中。当客户端再一次发出 JSP 请求时,就可以重用这个编译好的字节码文件,没有必要再把同一个 JSP 进行翻译和编译了,这就大大提高了 Web 应用系统的性能。当然,如果对 JSP 进行了修改,Web 容器就会及时地发现改变,此时 Web 容器就会重新执行翻译和编译(见图 3-3)。所以这也是我们访问 JSP 页面时第一次请求会比较慢,后续访问时速度就较快的原因。

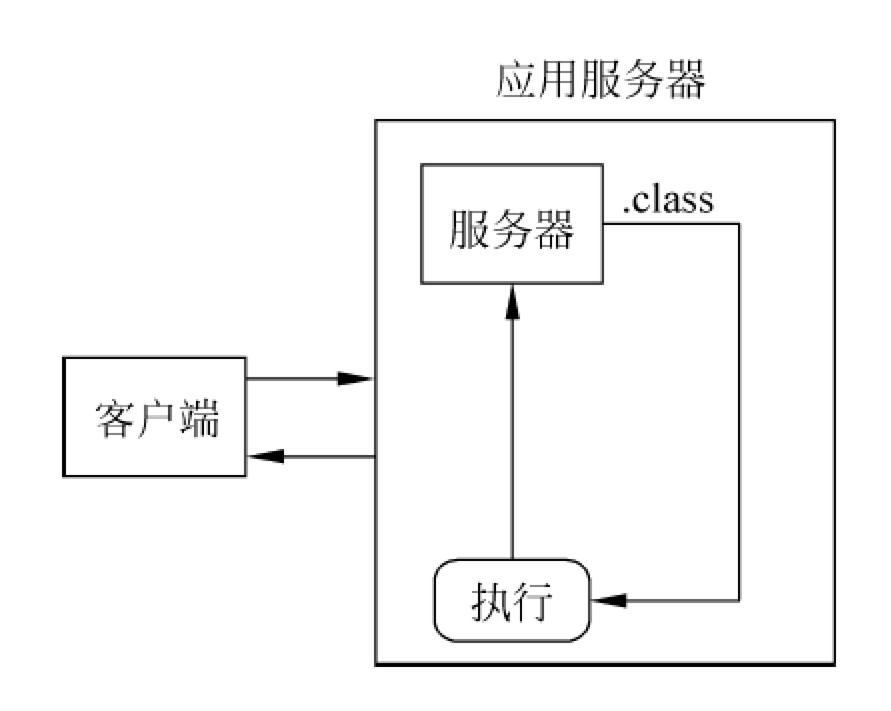


图 3-3　Web 容器处理 JSP 文件第二次请求

那么,我们已经了解了 JSP 的工作原理以及执行过程。这些是学习 JSP 的基础,而使用 JSP 实现动态网页开发,还要熟悉 JSP 页面里包含什么元素,不同元素具备什么功能。

前面已经谈到:JSP 是通过在 HTML 中嵌入 Java 脚本语言来响应页面动态请求。那么 JSP 页面的基本构成是什么呢?JSP 页面由静态内容、指令、表达式、小脚本、声明、标准动作和注释等元素构成。下面通过示例 3-1 展示几个比较常用的 JSP 页面元素。

示例3－1

```
/* *
 * jsp 页面源代码
 *
 */
<%@ page language = "java" import = "java.util.*,java.text.*" contentType = "text/html;
charset = UTF-8" %>
<html>
<head>
<title>输出当前日期</title>
<!-- 这是 HTML 注释(客户端可以看到源代码) -->
<%-- 这是 JSP 的注释(客户端不可以看到源代码)-- %>
</head>
<body>
你好,今天是
<%
    //使用预定格式将日期转换为字符串
    SimpleDateFormat formater = new SimpleDateFormat("yyyy 年 MM 月 dd 日");
    String strCurrentTime = formater.format(new Date());
%>
<% = strCurrentTime %>
<%!String str = "this is JSPPage"; %>
<% = str %>
</body>
</html>
```

在浏览器中显示示例 3-1 的运行结果如图 3-4 所示。

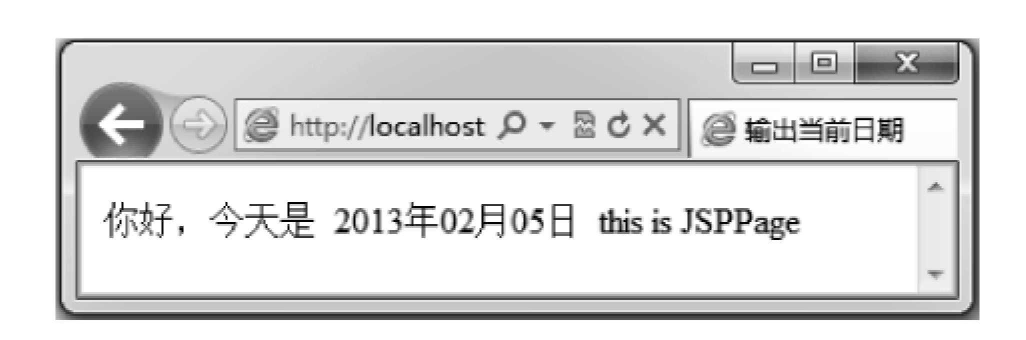


图 3-4　在浏览器上观察示例 3-1 的运行结果

示例产生的页面源代码如图 3-5 所示。

http://localhost:8080/Ch03_1/ - 原始源

文件(F) 编辑(E) 格式(O)

```

<html>
<head>
<title>输出当前日期</title>
<!-- 这是HTML注释（客户端可以看到源代码） -->

</head>
<body>
你好，今天是

2013年02月05日

this is JSPPage
</body>
</html>

```

图 3-5　查看示例 3-1 的页面源代码

在示例 3-1 中，一共展示了六种页面元素，包含静态内容、指令、小脚本、表达式、声明以及注释。下面一一加以介绍。

3.2 静态内容

静态内容是 JSP 页面中的静态文本，它基本上是 HTML 文本，而与 Java 和 JSP 语法无关，HTML 在第 2 章我们已做过详细介绍。

3.3 JSP 中的注释

在编写程序的时候，每个程序员都要养成写注释的好习惯，合理、详细的注释有利代码后期的维护和阅读。在 JSP 文件的编写过程中共有 3 种注释方法。

1. HTML 注释方法，其使用格式是：<! -- HTML 注释-->。其中的注释内容在客户端浏览器里查看源代码时，可以看到这些注释内容，如图 3-5 所示。这种注释方法是不安全的，而且会加大网络的传输负担。

2. JSP 注释标记，其使用格式是：<%--JSP 注释--%>。在客户端通过查看源代码时看不到注释的内容，如图 3-5 所示，安全性比较高。

3. JSP 脚本中使用注释。脚本就是嵌入到“<%”和“%>”标记之间的程序代码，使用的语言是 Java，因此在脚本中进行注释和在 Java 类中进行注释的方法一样。其使用格式是：<%//单行注释%>、<%/ * 多行注释 * / %>。

3.4 JSP 指令元素

JSP 指令元素的作用是通过设置指令中的属性在 JSP 运行时，控制 JSP 页面的某些特性。需要注意的是，不能错误地从字面意义上理解 JSP 指令元素是用来进行逻辑处理或者是用于产生输出代码的命令。

JSP 指令一般以“<%@”开始，以“%>”结束。主要有以下 3 种：

- page 指令
- include 指令
- taglib 指令

3.4.1 page 指令

page(页面)指令是 JSP 页面中使用频率最高的指令，主要用来设置整个 JSP 页面的相关属性，如网页内容的类型、网页的编码方式、网页需导入的包、发生异常时的处理方式等信息。该指令无论写在页面文件的什么位置，都对整个页面有效。但是在一般情况下写在 JSP 页面的第一行。

page 指令的语法格式：

```
<%@ page 属性名 1="属性值 1"...属性名 n="属性值 n" %>
```

书写时请注意区分大小写。page 指令一共有 13 个属性可以设置，其中最常用的属性请参考表 3-1。

表 3-1 page 指令的属性

属 性	说 明	示 例
pageEncoding	指定 JSP 页面的字符编码	pageEncoding = ""
contentType	指定输出内容的 MIME 类型及字符集	contentType= "text/html; charset= utf-8"
Import	导入 Java API 或自定义的类包，该属性可以在一个页面中多次指定	import= "java. sql.* " import= "java. sql.* ,com. bean.* "

可以在一条 page 指令中指定多个属性，也可以分开多条指定。但除 import 属性可以在一个页面中多次出现外，其余属性只能指定一次，不能重复出现。

示例 3-1 使用了 page 指令。

3.4.2 include 指令

include(包含)指令所实现的文件包含又称为"静态包含"，它用于通知 JSP 容器将指定的资源内容嵌入到该条指令出现的地方。所包含的资源可以是 JSP 文件、HTML 文件，或者是其他类型的文本文件。语法格式如下：

```
<%@ include file= "relativeURL" %>
```

其中"relativeURL "表示被包含的资源的文件相对路径(含文件名)。

例如，login. jsp 页面中有以下指令：

```
<%@ include file= "pub/top. jsp" %>
```

这时，若 login. jsp 位于 D:\workspace\Ch03_1\WebRoot\page 下，则 top. jsp 应该位于 D:\workspace\Ch03_1\WebRoot\page\pub 下，即相对的是当前文件的位置。若改写成：

```
<%@ include file= "/pub/top. jsp" %>
```

那么，此时的 pub 目录应该位于何处呢？

与刚才的例子不同的是，该路径是以"/"开头的。这里路径最前面的"/"代表的是整个应用的发布根目录，即 D:\workspace\Ch03_1\WebRoot，top. jsp 文件应该位于 D:\workspace\Ch03_1\WebRoot\pub 目录下。可见，采用后面这种写法时，资源的路径与包含 top. jsp 的主页面无关(这里为了方便描述，暂且把像 login. jsp 这样包含其他资源的页面称为"主页面")。因此，将主页面(如 login. jsp)转移至其他目录时，该页面中的该指令无须做任何更改。

示例3-2

include 指令的使用。

在页面 main. jsp 中使用 include 指令包含 top. jsp 的内容。

```
/* *
 * top.jsp 页面源代码,该文件 WebRoot\page\top.jsp
 *
 */
<img src = "kenan.jpg" />
```

```
/* *
 * main.jsp 页面源代码,该文件 WebRoot\page\main.jsp
 *
 */
<%@ page language = "java" import = "java.util.*,java.text.*" contentType = "text/html;
charset = UTF-8" %>
<html>
<head>
<title>include</title>
</head>
<body>
测试文字 1<br/>
<%@include file = "top.jsp" %><br/>
测试文字 2
</body>
</html>
```

当访问 main.jsp 时得到如图 3-6 所示的运行结果。



图 3-6 访问 main.jsp 的效果图

使用 include 指令时须注意:

- 被包含的页面若出现了设置页面编码的 page 指令,则主页面中所出现的编码设置必须与此完全相同,否则可能引起翻译时的错误。
- include 指令所实现的包含操作是在当前的 JSP 文件被转换为 Servlet 时进行的,不是在执行阶段进行的,因此,file 属性的值不能是变量,也不能以形如"head.jsp?aa=1"这种形式向被包含页面传递参数。

那么在 JSP 页面中什么情况下使用 include 指令呢?若一个网站的不同页面有相同的部分,例如页头、页脚部分是相同的,则可以将它们分别制作为独立的文件,而后在所需要的

页面中将它们包含进来。

3.4.3 taglib 指令

taglib(标签库)指令告诉JSP容器怎样找到标签描述的文件和标签库,以及在JSP页面引用这个标签时的前缀。

taglib 指令的语法格式:

```
<%@ taglib uri="标签库的 URI" prefix="标签的前缀" %>
```

例如:

```
<%@ taglib prefix= "c" uri = " http://java.sun.com/jsp/jstl/core" %>
```

不能使用 JSP,jspx,java,javax,servlet,sun,sunw 等作为前缀。

3.5 JSP 脚本元素

在JSP页面中,将表达式(expression)、小脚本(scriptlet)、声明(declaration)统称为JSP脚本元素,用于在JSP页面中嵌入Java代码,实现页面的动态请求。

下面,我们就来进一步学习这三种元素,掌握它们的使用方法。

3.5.1 小脚本

小脚本可以包含任意的Java片断,形式比较灵活,通过在JSP页面中编写小脚本可执行复杂的操作和业务处理,编写方法就是将Java程序片断插入到“<% %>”标记中。

示例3-1中,属于小脚本的代码片断是:

```
<%
    //使用预定格式将日期转换为字符串
    SimpleDateFormat formater = new SimpleDateFormat("yyyy 年 MM 月 dd 日");
    String strCurrentTime = formater.format(new Date());
%>
```

现在我们就来使用小脚本实现一个简单的业务处理,循环输出数组中的数值。代码如示例3-3所示。

示例3-3

```
/**
 * jsp 页面循环输出数组的数值
 *
 */
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>循环输出数组的数值</title>
</head>
<body>
```

```
<%
    int[] value = { 60, 70 ,80 };
    for (int i = 0;i < value.length; i++){
        out.println(value[i]);
 %>
</body>
</html>
```

这段代码中使用到了 JSP 的一个隐式对象 out(隐式对象在下一章将要介绍),目前只需了解 out.println 方法是用来在页面中输出数据的就可以了。

运行示例 3-3 会发现有错误,在示例 3-3 中,出现了一个初学者非常易犯的错误,那就是 for 循环缺少了一个“}”号。正确的代码如示例 3-4 所示。

示例3-4

```
/**
 * jsp 页面循环输出数组的数值
 *
 */
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>循环输出数组的数值</title>
</head>
<body>
<%
    int[] value = { 60, 70 ,80 };
    for (int i = 0;i < value.length; i++){
        out.println(value[i]);
 %>
    <br/>
<%
    }
 %>
</body>
</html>
```

效果如图 3-7 所示。

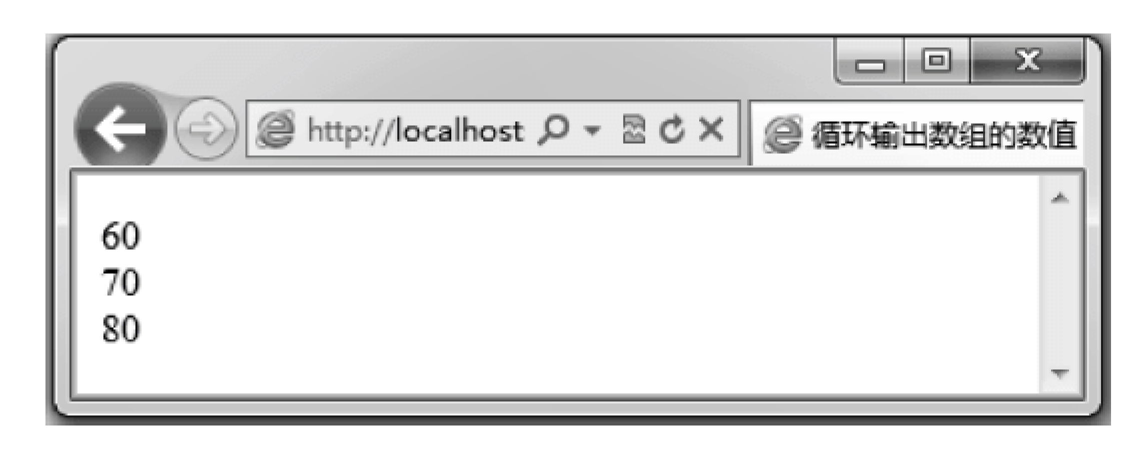


图 3-7　小脚本循环输出数组中的数

由于小脚本与 HTML 的混合编写，这种错误既容易发生，又不易被发现。这就需要在编写代码时不仅要注意代码的缩进，编写必要的注释，还要有足够的细心。

3.5.2 表达式

表达式是对数据的表示，系统将其作为一个值进行计算和显示。当需要在页面中获取一个 Java 变量或者表达式值时，使用表达式是非常方便的。其语法是<%＝Java 表达式%>。当 Web 容器遇到表达式时，会先计算嵌入的表达式值或者变量值，然后将计算结果以字符串形式返回并插入到相应的页面中。

下面，我们就修改一下示例 3-4 的代码，使用表达式实现数据的输出，代码如示例 3-5 所示。

示例3-5

```
/**
 * jsp 页面循环输出数组的数值
 *
 */
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<html>
<head>
<title>循环输出数组的数值</title>
</head>
<body>
<%
    int[] value = { 60, 70 ,80 };
    for (int i = 0;i<value.length; i++){
            %>
            <%= value[i]%><br/>
    <%
        }
    %>
</body>
</html>
```

需要注意的是，在 Java 语法的规定中，每一条语句末尾必须要使用分号代表结束。而在 JSP 中，使用表达式输出显示数据时，则不能在表达式结尾处添加分号，代表语句结束。这一点需要在代码编写过程中加以区分。



资料

在 Java 开发过程中，我们使用 System. out. println()和 System. out. print()向控制台输出信息。在 JSP 网页上，我们可以使用它的内置对象 out 把结果输出到页面上，通常最常使用的是 println (String message)和 print (String message)两个方法，当然也可以使用表达式实现输出信息的效果。

3.5.3 声明

在编写 JSP 页面程序时，有时需要为 Java 脚本定义变量和方法，这时就需要对所使用的变量或者方法进行声明。

声明的语法如下。

```
<%! Declaration; [ Declaration;]... %>
```

例如：

```
<%! int a = 1; %>
<%! String name; %>
```

JSP 语法中，变量和方法必须要先声明，否则就会出错，可以一次声明多个变量和方法，只要以"；"结尾即可，而且必须保证这些声明在 Java 中是合法的。

需要注意，声明与小脚本和表达式除了语法格式上的不同，还有就是声明一般不会有输出，通常与表达式、小脚本一起综合运用。下面我们就通过一个声明方法的示例来具体了解声明的作用。

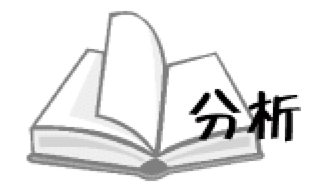
首先编写好如下一段代码，用于对日期进行格式化。

```
<%
SimpleDateFormat formater = new SimpleDateFormat("yyyy年MM月dd日");
String strCurrentTime = formater.format(new Date());
%>
```

现在有一个问题需要我们去解决，看下面的问题。

问题

在同一个 JSP 页面中，如果需要在多个地方格式化日期，如何简化代码？



分析

在 Java 文件中，可以增加一个方法来解决。在 JSP 文件中，同样可以声明方法解决类似问题。

具体代码如示例 3-6 所示，其运行结果如图 3-8 所示。

示例3-6

```
<!--/**
 * jsp声明的应用
 *
 */
--><%@ page language="java" import="java.util.*,java.text.*" contentType="text/
html; charset=UTF-8" %>
<html>
<head>
<title>方法声明</title>
```

```
</head>
<body>
<%!
    String formatDate(Date d){
        java.text.SimpleDateFormat formater = new SimpleDateFormat("yyyy年MM月dd日");
        return formater.format(d);
    }
%>
    第一次调用: 今天是<% = formatDate(new Date()) %>
    <br>
    第二次调用: 今天是<% = formatDate(new Date()) %>
</body>
</html>
```

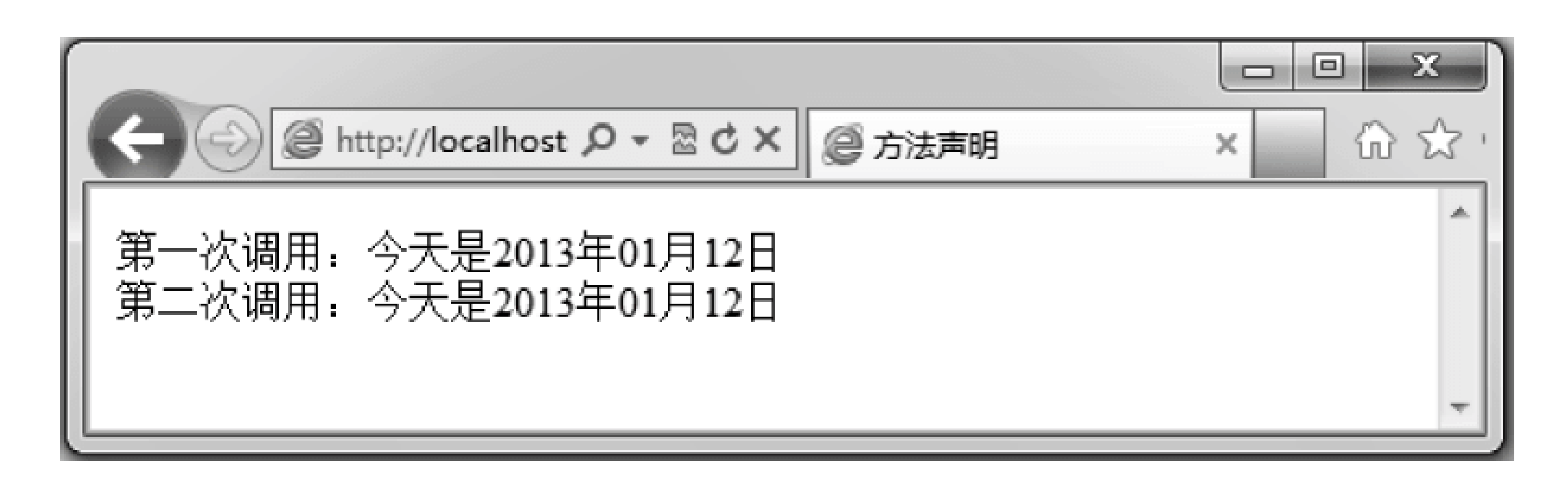


图 3-8 示例 3-6 的效果显示

3.6 JSP 动作元素

JSP 动作元素(Action Element)和 JSP 指令元素不同,它是在客户端请求时动态执行的。JSP 动作元素是一种特殊标签,并且以前缀 jsp 和其他的 HTML 标签相区别,利用 JSP 动作元素可以实现很多功能,包括动态地插入文件、重用 JavaBean 组件、把用户重定向到另外的页面、为 Java 插件生成 HTML 代码等。

动作元素包括<jsp:include>、<jsp:plugin>、<jsp:forward>、<jsp:param>、<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>等。其中后三个动作元素与 JavaBean 结合得非常紧密,将在第 5 章中详细说明。

3.6.1 <jsp:param>

<jsp:param>动作元素主要用来传递参数给 JSP 程序,而由程序获取的参数值,在程序中就是一个变量值。此操作元素的语法如下:

```
<jsp:param name = "attributeName" value = "attributeValue"/>
```

其中 name 属性表示传递参数的名称,value 属性是用来设置该参数的值。

在 JSP 中如何来获取<jsp:param>设定的参数值呢? 是通过内置对象 request 的 getParameter()方法,即 request. getParameter(attributeName)(内置对象在第 6 章介绍)。

<jsp:param>元素必须配合<jsp:include>、<jsp:forward>及<jsp:plugin>等使用。

3.6.2 <jsp:include>

<jsp:include>动作元素可以用来包含其他静态和动态页面。<jsp:include>有带参数和不带参数两种语法格式，分别是：

```
<jsp:include page="relativeURL" flush="true|false">
```

和

```
<jsp:include page="relativeURL" flush="true|false">
    <jsp:param name="attributeName1" value="attributeValue1"/>
    <jsp:param name="attributeName2" value="attributeValue2"/>
…
</jsp:include>
```

其中，relativeURL 指代被包含文件的相对路径；属性 flush 为 true 时，表示实时输出缓冲区，一般情况下 flush 都为 true。一个<jsp:include>中可以包含一个或多个<jsp:param>动作元素。下面就通过示例 3-7 来了解<jsp:include>的用法。

示例3-7

main.jsp

```
<!--/**
 * jsp:include 带参数引用
 *
 */
 -->
<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>include 带参数引用</title>
</head>
<body>
<jsp:include page="param.jsp" flush="true">
    <jsp:param name="str1" value="Hello" />
    <jsp:param name="str2" value=" JSP!" />
</jsp:include>
</body>
</html>
```

param.jsp

```
<!--/**
 * jsp:include 带参数引用
 *
 */
 -->
<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>
```

```
<%
    String str1 = request.getParameter("str1");
    String str2 = request.getParameter("str2");
    out.print(str1);
    out.print(str2);
%>
```

示例 3-7 访问 main.jsp 页面的效果显示如图 3-9 所示。

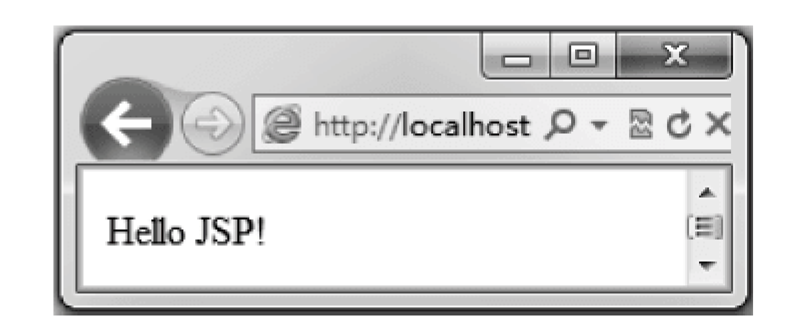
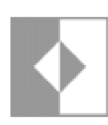


图 3-9 示例 3-7 的效果显示

JSP 中有两种不同的包含方式：编译时包含和运行时包含。编译时包含只是将静态文件内容加到 JSP 页面中，优点是速度快，例如<%@ include%>指令包含。运行时包含指的是被包含的文件运行时被 JSP 容器编译执行，<jsp:include>的包含就是运行时包含，当然同时支持编译时包含。


比较

JSP 中 include 指令 与<jsp:include>的区别

include 指令是指把其他页面的代码加进来，与本页面的代码合并在一起，相当于把源代码从那个页面中复制到本页面来，然后再编译。由于本页页面编译时已经包含别的文件的源代码，所以以后其他页面的代码更改时，本页面并不理会，因为已经编译过了。这种包含方式为静态包含。

<jsp:include>动作是指两个页面的代码运行完以后，再把包含的那个页面运行后的 HTML 结果页面，加到本页面运行后的 HTML 结果页面中来。所以是运行时包含，并且还可以传递参数给被包含的页面。这种包含为动态包含。

3.6.3 <jsp:forward>

<jsp:forward>用于服务器端结束当前的页面，并从当前页面跳转到其他指定页面(<jsp:forward>为转发而不是重定向)。转向的目标页面可以是 HTML 页面、JSP 文件或 Servlet 类。

<jsp:forward>也有带参数和不带参数两种语法格式，分别是：

```
<jsp:forward page="pageURL">
```

和

```
<jsp:forward page=" pageURL">
    <jsp:param name="attributeName1" value="attributeValue1"/>
    <jsp:param name="attributeName2" value="attributeValue2"/>
    …
</jsp:forward>
```

<jsp:forward>最典型的应用就是登录时进行权限验证，验证通过，就把页面 forward 到登录成功页面；当验证通不过时，则把页面 forward 到登录页面。

示例3-8

login.jsp

```
<!--/**
 * 登录页面
 */
-->
<%@ page language="java" import="java.util.*,java.text.*" contentType="text/html;
charset=UTF-8"%>
<html>
<head>
<title>登录页面</title>
</head>
<body>
<form action="checkLogin.jsp" method="post">
用户名:<input type="text" name="name"><br/>
密 码:<input type="password" name="password">
<input type="submit" value="登录">
</form>
</body>
</html>
```

checkLogin.jsp

```
<!--/**
 * 验证登录页面
 */
-->
<%
    String name = request.getParameter("name");
    String password = request.getParameter("password");
    if(name.equals("admin")&&password.equals("123")){
        %>
            <jsp:forward page="success.jsp">
                <jsp:param name="name" value="<%=name %>" />
            </jsp:forward>
        <%
    }
    else {
        %>
            <jsp:forward page="login.jsp"/>
        <%
    }
%>
```

success. jsp

```
<! -- / * *
 * 登录成功页面
 * /
 -- >
<% @ page language = "java" import = "java. util. * , java. text. * " contentType = "text/html;
charset = UTF - 8" %>
<html>
<head>
<title>登录成功</title>
</head>
<body>
登录成功!<br/>
欢迎你,<% = request. getParameter("name") %>
</body>
</html>
```

login. jsp 的执行结果如图 3-10 及图 3-11 所示,输入正确的用户名、密码后单击“登录”按钮提交,经 checkLogin. jsp 的验证,成功则 forward 到 success. jsp 页面,并把参数传给 success. jsp 页面,失败则 forward 到 login. jsp 页面。

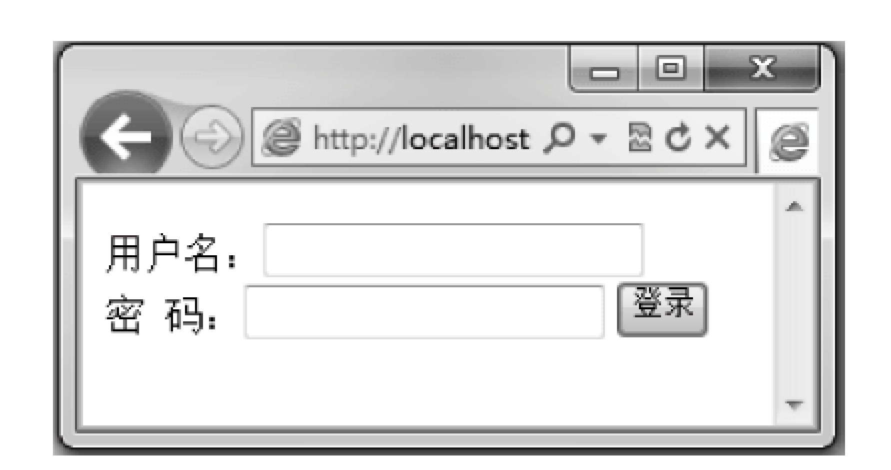


图 3-10 登录即登录失败后页面

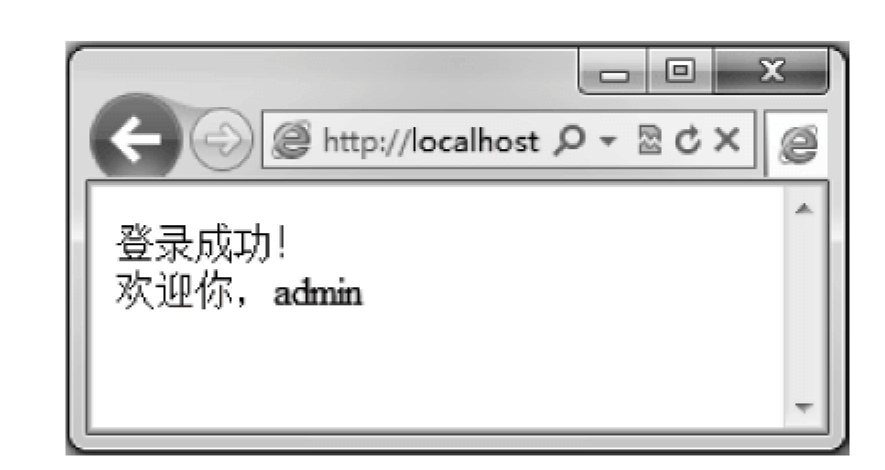


图 3-11 登录成功后页面

3. 6. 4 <jsp:plugin>

<jsp:plugin>可以在页面中插入 JavaApplet 小程序或 JavaBean,它们能在客户端运行。其语法格式如下:

```
<jsp:plugin
type = "bean | applet"
code = "classFileName"
codebase = "classFileDirectoryName"
[ name = "instanceName" ]
[ archive = "URIToArchive, ..." ]
[ align = "bottom | top | middle | left | right" ]
[ height = "displayPixels" ]
[ width = "displayPixels" ]
[ hspace = "leftRightPixels" ]
[ vspace = "topBottomPixels" ]
[ jreversion = "JREVersionNumber | 1.1" ]
```

```
[ nspluginurl = "URLToPlugin" ]
[ iepluginurl = "URLToPlugin" ] >
[ < jsp:params >
[ < jsp:param name = "parameterName" value = "{parameterValue | < % = expression % >}" /> ] +
</jsp:params > ]
[ < jsp:fallback > text message for user </jsp:fallback > ]
</jsp:plugin >
```

<jsp:plugin>元素的属性如表 3-2 所示。

表 3-2 <jsp:plugin>的属性及说明

属 性	说 明
type	加载 Java 程序的类型，可设置的值有 applet 及 bean。其中 applet 代表加载 Java Applet 程序，bean 则代表加载 JavaBean
code	指定要加载的 Java 的类文件的名称，必须以.class 结尾命名
codebase	指定将会执行的 Java 类文件所在的目录或路径，默认调用当前访问的 JSP 文件的路径
name	指定了加载的 Applet 或 Bean 的名称
align	加载的插件对象在页面中的对齐方式，可选的值有：bottom，top，middle，left，right
height 和 width	设置加载对象的高度和宽度，单位为像素
hspace 和 vspace	加载的插件与页面其他内容的水平间隔和垂直间隔
<jsp:param>	向 Applet 或 Bean 中传递参数
<jsp:fallback>	该指令中间的一段文字用于 Java 插件不能启动时显示给客户端，如果插件能够正确启动，而 JavaBean 或 Java Applet 的程序代码不能找到并执行，那么浏览器将会显示这个错误信息

示例3-9

```
< jsp:plugin >
type = "applet"
code = "Test.class"
codebase = "/example/jsp/applet"
height = "180"
width = "200"
< jsp:params >
< jsp:param name = "test" value = "TestPlugin"/>
< jsp:fallback > To load applet is unsuccessful </jsp:fallback >
</jsp:plugin >
```

3.7 上 机 练 习

1．编写 JSP 页面 index.jsp 、login.jsp、backIndex.jsp。

需求说明：通知公告发布系统的前台页面、登录页面、后台页面由 HTML 改为 JSP 页面，如图 3-12～图 3-14 所示。



图 3-12 通知公告发布系统 index.jsp 页面效果图



图 3-13 通知公告发布系统后台 login.jsp 页面效果图

实现思路：

(1) 将页面修改成 JSP。

(2) 修改链接，将.html 改为.jsp。

(3) 启动服务器，浏览 JSP 页面，检查链接和表单验证。

提示修改 JSP 的方式：

方式 1：使用向导创建 JSP，复制静态内容。

方式 2：将.htmL 文件名的后缀修改为.jsp，在 JSP 首行加入：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8" %>
```

2. 添加注释。

需求说明：在练习 1 中的 jsp 中添加 HTML 注释、JSP 注释、JSP 脚本注释。

实现思路：

```
提示: <!-- HTML 注释 -->
```



图 3-14 通知公告发布系统后台 backIndex.jsp 页面效果图

```
<% --JSP 注释 -- %>
<% //JSP 脚本注释 %>
<% /* JSP 脚本注释 */ %>
```

3. 使用脚本实现简单的动态输出。

需求说明：在通知公告发布系统的 index.jsp 页面中添加动态输出日期、时间的功能

实现思路：

在 index.jsp 页面中添加 JSP 脚本关键代码如下：

```
<%@ page language="java" import="java.util.*,com.bean.*" pageEncoding="UTF-8"%>

......
<%
//使用预定格式将日期转换为字符串
java.text.SimpleDateFormat formater = new
java.text.SimpleDateFormat("yyyy年MM月dd日");
String strCurrentTime = formater.format(new Date());
%>
<%= strCurrentTime %>
......
```

完成后效果如图 3-15 所示。



图 3-15 增加了动态显示时间 index.jsp 页面效果图

3.8 总　　结

(1) JSP 的工作原理，当用户通过浏览器请求访问 Web 应用时，Web 服务器会使用 JSP 引擎对请求的 JSP 进行编译和执行，然后将生成的页面返回给客户端浏览器进行显示。Web 容器通过翻译阶段、编译阶段、执行阶段三个阶段对 JSP 进行处理。

(2) JSP 文件的编写过程中共有三种注释方法

HTML 注释方法，格式是：<! -- HTML 注释-->；JSP 注释标记，格式是：<%--JSP 注释--%>；JSP 脚本中使用注释，格式为<%//单行注释%>、<%/ * 多行注释 * / %>。

(3) JSP 的常用指令元素有三种：page、include、taglib。

(4) 在 JSP 中，将表达式(expression)、小脚本(scriptlet)、声明(declaration)统称为 JSP 脚本元素。

(5) JSP 中动作元素包括<jsp:include>、<jsp:plugin>、<jsp:forward>、<jsp:param>、<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>等。

3.9 作　　业

一、选择题

1. 以下第(　　)种注释可以被发送到客户端的浏览器。

```
<%-- 第一种 --%>
<% //第二种 %>
<% /* 第三种 */ %>
<! -- 第四种 -->
```

A. 第一种　　B. 第二种　　C. 第三种　　D. 第四种

2. 在某个 JSP 页面中存在这样一行代码：<%="2"+"4"%>，运行该 JSP 后，以下说法正确的是(　　)。

A. 这行代码对应的输出是 6　　B. 这行代码对应的输出是 24

C. 这行代码没有输出　　D. 这行代码将引发错误

3. 与 page 指令<%@ page import="java.util.*,java.text.*"%>等价的是(　　)。

A. <%@ page import="java.util.*"%>
   <%@ page import="java.text.*"%>

B. <%@ page import="java.util.*" import=" java.text.*"%>

C. <%@ page import="java.util.*,"; %>
   <%@ page import="java.text.*"; %>

D. <%@ page import="java.util.*;java.text.*"%>

4. 如果请求页面中存在两个单选按钮(假设单选按钮的名称为 is)。分别代表是和否，该页面提交后，为了获得用户的选择项，可以使用以下(　　)方法。

A. request.getPararmeter(is)　　B. request.getPararmeter("is")

C. request.getPararmeterValues(is)　　D. request.getPararmeterValues("is")

5. page 指令用于定义 JSP 文件中的全局属性，下列关于该指令用法的描述中错误的是(　　)。

A. <%@ page %>作用于整个 JSP 页面

B. 可以在一个页面中使用多个<%@ page %>指令

C. 为增强程序的可读性，建议将<%@ page %>指令放在 JSP 文件的开头，但不是必需的

D. <%@ page %>指令中的属性只能出现一次

二、简答题

1. Web 处理 JSP 请求的三个阶段是什么？

2. 给定如下 JSP 代码，请说明其中包含了哪些 JSP 页面元素。

```
<%@ page language="java" import="java.util.*" contentType="text/html; charset=UTF-
8"%>
<html>
<head>
<title>输出字符</title>
<%-- 这是输出字符的代码 --%>
</head>
<body>
<%
    String str = "hello JSP";
```

```
 %>
<% = str %>
</body>
</html>
```

3. JSP 的动作元素包括哪几种？

三、程序题

编写一个 JSP 页面，在两个文本框中输如两个数字，提交后比较两个数的大小，输出其中较大的值。

第 4 章　JSP 数据库应用开发

本章学习目标

- 了解数据库的基本概念
- 了解数据库 SQL Server 2008
- 掌握在 SQL Server 2008 中创建数据库
- 掌握在 SQL Server 2008 中创建表
- 熟悉 SQL 的常用查询语句
- 了解 JDBC 的工作原理
- 掌握使用 JDBC 连接数据库的方法
- 了解 JDBC-ODBC 桥连
- 掌握纯 Java 驱动方式连接数据库
- 掌握在 WEB 项目中创建连接数据库的工具类

JSP 的数据库应用开发是 JSP 开发中的重点内容，多数的 Web 应用都离不开 JSP 与数据库的结合。JSP 的数据开发技术可以根据不同的数据呈现不同的页面，比如本书的贯穿项目通知公告发布系统等。在 JSP 中的数据库编程主要是通过 JDBC 来实现的。本章就结合案例来介绍 JSP 数据库应用开发方面的知识。

4.1　数据库简介

数据库技术产生于 20 世纪 60 年代末，多年来数据库技术得到了迅速的发展，已形成了较为完整的理论体系和一大批实用系统，数据库技术已融入到金融、商业和工程技术等各个领域。

4.1.1　数据库基本术语

数据(Data)是对客观事物描述和记载的可以鉴别的物理符号，是客观事物的基本表达。与数据相对应的是信息，信息是数据的集合、含义与解释，是事物变化、特征的反映。

数据库(DataBase，DB)是指在计算机内按一定形式存放、有组织、统一管理的相关数据和数据库对象的集合。数据库对象是指表(Table)、视图(View)、存储过程(Stored Procedure)、触发器(Trigger)等。

数据库管理系统(DataBase Management System,DBMS)是位于用户和操作系统(Operation System,OS)之间的一层数据管理软件,它能够科学地组织和存储数据、高效地获取和维护数据,并能为用户提供访问数据库的方法,包括数据库的建立、查询、插入、修改和删除等。

数据库管理员(DataBase Administrator,DBA)是指负责数据库系统正常运行、承担数据库的创建、监控、维护等工作的一组人员。数据库管理员主要有以下职责:决定数据库中的信息内容、存储结构、存储策略等,定义数据库的安全性要求和完整性约束条件,监控数据库的使用和运行,包括数据库的转储、故障恢复等,数据库的优化、重组、重构等。

数据库系统(DataBase System,DBS)是指实现有组织、动态地存储大量关联数据、方便用户访问的计算机硬件、软件和数据资源的系统,它主要由数据库、数据库管理系统、应用系统、数据库管理员及用户组成。在不引起混淆的情况下,常将数据库系统简称为数据库,如图4-1所示。

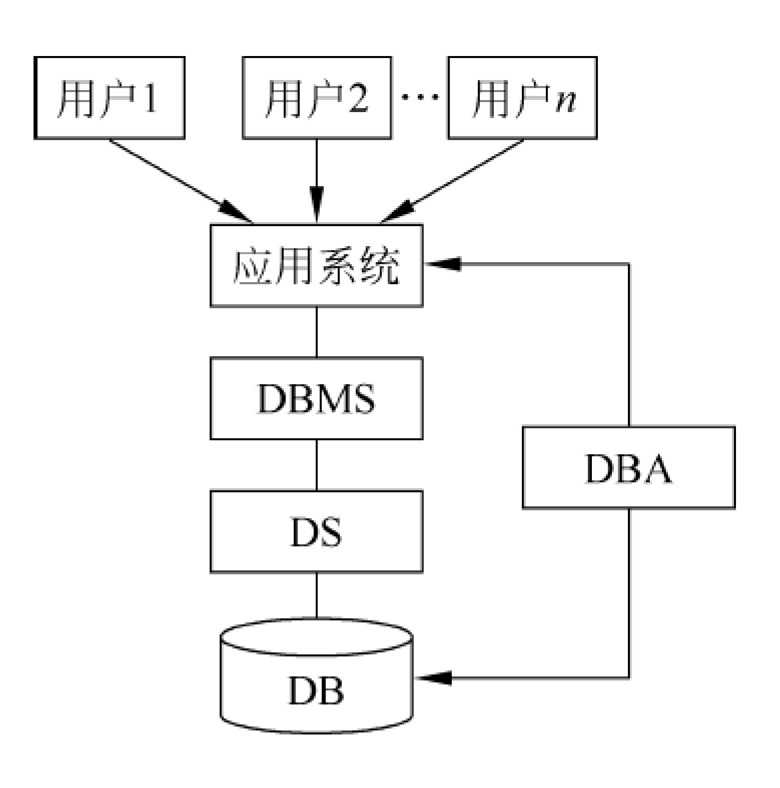


图 4-1 数据库管理系统(DBMS)

4.1.2 关系数据库

根据数据库管理系统基于的数据模型,可将数据库分为层次数据库、网状数据库、关系数据库、面向对象数据库。目前应用最广泛的是关系数据库,支持关系数据库的数据库产品有很多,如IBMDB2、Oracle、Sybase、Microsoft SQL Server等。本节主要介绍关系数据库的一些基础知识,包括关系数据库的基本概念、完整性规则等内容。

1. 基本概念

关系数据库是以关系模型为基础的,关系模型是利用二维表格表示数据的数据模型。下面以贯穿项目的notice数据库为例,介绍关系数据库中的基本概念。

数据库notice中保存用户信息、通知公告信息、通知公告类型信息,分别见表4-1、表4-2、表4-3。由于表4-1、表4-2、表4-3关系模型与二维表格类似,因此采用关系模型来表示notice数据库,并将notice数据库中存储的三个数据表(DataTable)命名为Notice、Nuser和Type,也称为关系Notice、关系Nuser和关系Type。

表 4-1 用户表

Uno	Uname	Upassword
1	admin	admin
2	a	a

表 4-2 通知公告信息表

Nno	Ntitle	Ncontent	Neditor	NcreateTime	Ntype
1	放假通知	元旦放假通知	教务处	2012-12-26	1
2	录入成绩通知	请考试结束的	教务处	2012-12-20	1
3	招标公告	我校实验中心	设备处	2012-11-15	3

表 4-3 通知公告类型表

Tno	TtypeName
1	教学通知
2	科研通知
3	招标公告

关系的首行称为属性(atrribute),也称为字段(field)等,关系的属性就是关系各列的名字,属性描述了所在列的意义。例如关系 Nuser 中具有如下 3 个属性:Uno、Uname 和 Upassword,各个属性分别表示用户的编号、姓名和密码。

关系中每一个属性都有一个取值范围,称为该属性的域(Domain)。例如在通知公告信息关系中,属性 Ntitle、Ncontent、Neditor 的域必须是字符串型,属性 NcreateTime 的取值必须为日期类型,属性 Ntype 必须为整数型。

关系名和关系的属性集合称为关系的模式。要表示一个关系的模式,一般用括号将属性集括起来,并将关系名写在括号的前面,格式如下:

```
Relation_Name(attribute1,attribute2,attribute3,…)
```

下面表示关系 Nuser 的模式:

```
Nuser(Uno,Uname,Upassword)
```

在关系中,字段的有序集合称为记录,记录的各个分量分别对应着关系的各个属性。要表示一条记录,一般用括号将整条记录的分量括起来,并用逗号将各分量隔开,如下表示一条记录:

```
(1,admin,admin)
```

在关系中,一条记录中有若干个属性,若其中某一个属性组能唯一标识一条记录,该属性组就可以成为一个主键。

例如,在关系 Nuser(Uno,Uname,Upassword)中,用户的编号是唯一的,也可以标识一条记录,所以 Uno 就是一个主键。在关系 Notice(Nno,Ntitle,Ncontent,Neditor,Ncreate Time,Ntype)中,notice 的编号也是唯一的,也可以标识一条 notice 记录,所以 Nno 就是关系 Notice 的主键,同理,Tno 就是关系 Type(Tno,TtypeName)的主键。

关系 Notice 中 Ntype 不是主键,但是它和关系 Type 中的 Tno 对应,并且 Tno 是关系 Type 的主键,则称关系 Notice 中的 Ntype 是关系 Notice 的外键。

总结

定义主键和外键主要是为了维护关系数据库的完整性。

主键是能确定一条记录的唯一标识,例如,一条记录包括学号、姓名、年龄。学号是唯一能确定这个人的,其他都可能有重复,所以,学号是主键。外键用于与另一张表的关联,是能确定另一张表记录的字段,用于保持数据的一致性。比如,A 表中的一个字段,是 B 表的主键,那它就可以是 A 表的外键,则 A 表中该字段的取值,应在 B 表主键取值的范围中。

数据库表结构是指这个数据库表名字、字段、主键等表的信息。这三个数据表的表结构如表 4-4~表 4-6 所示。

表 4-4 Nuser 表结构

字段名称	数据类型	是否主键	说　明
Uno	int	是	自动编号
Uname	nvarchar(50)		用户姓名
Upassword	nvarchar(50)		用户密码

表 4-5 Notice 表结构

字段名称	数据类型	是否主键	说　明
Nno	int	是	自动编号
Ntitle	nvarchar(200)		通知公告标题
Ncontent	ntext		通知公告内容
Neditor	nvarchar(50)		编辑人
NcreateTime	datetime		创建时间
Ntype	int	外键	类型

表 4-6 Type 表结构

字段名称	数据类型	是否主键	说　明
Tno	int	是	自动编号
TtypeName	nvarchar(100)		通知公告类型名

2. 完整性规则

为了使数据库中的数据和现实世界保持一致，关系数据库定义了如下三类完整性规则：实体完整性、参照完整性和用户定义的完整性。

(1) 实体完整性规则(Entity Integrity Rule)

在该规则中，若属性集 A 为关系 R 的主键，则属性集 A 不能取空值，否则主键值就不能起到唯一表示记录的作用。例如在关系 Nuser 中，属性 Uno 为关系 Nuser 的主键，则该属性对应的值不能为空值。

(2) 参照完整性规则(Reference Integrity Rule)

该规则规定，若属性集 F 为关系 R1 的主键，且属性集 F 为关系 R2 的外键，则在关系 R_2 中属性集 F 的取值必须是以下两种情况之一：空值(属性集 F 中的每个属性均为空值)或等于关系 R1 中某个主键值。例如在关系 Notice 和 Type 中，属性 Ntype 为关系 Notice 的外键，则属性 Ntype 在关系 Notice 中的取值只能为空值或等于关系 Type 中属性 Tno 的某个值。

(3) 用户定义的完整性规则

用户定义的完整性规则是针对某一具体关系数据库的约束条件，它反映某一具体应用所涉及的数据必须满足的语义要求。例如在某关系中，人的年龄可以利用定义的完整性规则，将年龄限定在 0～200 之间。

4.2 结构化查询语言 SQL 简介

SQL (Structured Query Language)是结构化查询语言的简称。SQL 的主要功能是同各种数据库建立联系,进行沟通,它可用来执行各种操作,如数据库中检索数据、更新数据库中的数据等。

目前绝大多数的关系数据库管理系统如 IBM DB2、Microsoft SQL Server、MySQL 等都采用了 SQL 标准,这些数据库系统除了采用 SQL 标准外,也对 SQL 进行了扩展,然而一些常用的标准 SQL 命令,如 select、insert、update、delete、drop、create 等,几乎可以完成所有的数据库的操作。

4.2.1 SQL 的组成

按照 SQL 的功能,SQL 由以下 3 部分组成。

(1) 数据定义语言(Data Definition Language,DDL) 它主要用于定义 SQL 模式、数据表、视图和索引等结构。

(2) 数据操纵语言(Data Manipulation Language,DML) 它可分为数据查询和数据更新两类,其中数据更新又可分为数据插入、数据删除和数据修改。

(3) 数据控制语言(Data Control Language,DCL) 它用来设定或更改数据库用户或角色,包括对数据表和视图的授权、完整性规则的描述、事务控制等。

在上述 SQL 中,JSP 最常用的为数据定义语言和数据操纵语言。下面就来介绍 SQL 中最常用的命令。

4.2.2 SQL 中常用的命令

1. 创建数据库 create database

在 SQL 中,创建一个新数据库的基本语法格式如下:

```
create database 数据库名称
```

例如:create database notice。

数据库名称在服务器中必须唯一,并且符合标识符的命名规则。

2. 创建表 create table

在 SQL 中,创建一个新表的基本语法格式如下:

```
create table 表名称(列名数据类型,…)
```

例如:

```
create table Nuser(Uno int,Uname nvarchar(50),Upassword nvarchar(50)).
```

表的名称必须符合标识符命名规则,列名即为字段名,必须符合标识符规则,并且在表内唯一。数据类型可以是系统数据类型或用户定义数据类型。

在数据库中,为了操作和编程方便,表名和字段名一般用代号表示。例如,对于关系模式用户信息(编号,姓名,密码)的表,用 Nuser 表示用户表名,用 Uno,Uname 和 Upassword 分别表示各项目的字段名。同样,通知公告信息表:表名 Notice,编号、标题、内容、编辑人、

创建时间和类型，分别对应的字段为 Nno、Ntitle、Ncontent、Neditor、Ncreatetime 和 Ntype。通知公告类型表：表名 Type，编号、类型名称对应的字段名分别为 Tno 和 TtypeName。

3. 插入数据语句 insert

insert 可添加记录到表中，其语法形式如下：

```
insert into 表名[(字段名表)] values(值表)
```

例如，向用户表添加一条记录，并给所有字段赋值：

```
insert into Nuser values('rose','red');
```

例如，向 Notice 表添加一条记录，并给 3 个字段赋值：

```
insert into Notice (Ntitle,Ncontent,Neditor) values('放假通知','元旦放假三天','教务处')
```


说 明

SQL 中字母是不区分大小写的，语法格式中所出现符号的意义约定如下：

“|”分隔括号或大括号中的语法项，只能选择其中一项；

“[ ]”可选语法项(不要输入方括号)；

“{ }”必选语法项(不要输入大括号)；

“[…n]”指示前面的项可以重复 n 次，每一项由逗号分隔；

“( )”一定要输入小括号。

4. 删除数据语句 delete

delete 用来从表中删除记录，其语法格式如下：

```
delete from 表名[where 条件]
```

例如，从 Nuser 表中删除姓名为"rose"的记录：

```
delete from Nuser where Uname = 'rose'
```

5. 更新数据语句 update

update 语句用来更新表中的记录，其语法格式如下：

```
update 表名 set 字段名 1 = 值 1[,字段名 2 = 值 2… ][where 条件]
```

例如，将 Notice 表中的 Neditor 改为'教务处'条件是 Neditor 为'财务处'

```
update notice set Neditor  = '教务处' where Neditor = '财务处'
```

6. 数据查询语句 select

select 的语法形式如下：

```
select [distinct][别名.]字段名或表达式[as 列标题]
```

from 表或视图别名

```
[where 条件]
[groupby 分组表达式]
```

```
[orderby 排序表达式[ asc|desc ]]
```

其中：select 子句指出查询结果中显示的字段名或字段名和函数组成的表达式等，as 列标题指定查询结果显示的列标题。若要显示表中所有字段，可用通配符“*”代替字段名列表。可用 distinct 去除重复的记录行；from 子句指定表或视图；where 子句定义了查询条件；groupby 子句对查询结果分组；orderby 子句对查询结果排序。语法形式中的“别名”，是给表另起一个简单的名字，以供调用其属性时使用。

例如以下查询内容。

（1）查询 notice 数据库中的 Notice 表中的标题和内容

```
use notice
select Ntitle,Ncontent from Notice
```

（2）查询 Notice 表中所有的信息

```
select * from Notice
```

（3）查询 Nuser 表中信息 Uname 显示名为“用户名”，Upassword 显示名为“密码”

```
select u.Uname as 用户名,u.Upassword as 密码 from Nuser u
```

（4）查询 Notice 表中 Nno 为 1 的信息

```
select * from Notice where Nno = 1
```

（5）查询 Notice 表中标题中有“放假”的信息

```
select * from Notice where Ntitle like '%放假%'
```

（6）查询 Notice 表中由教务处发通知的信息

```
select * from Notice where Neditor = '教务处'
```

（7）查询 Notice 表中发布时间在 2012-12-01 到 2013-01-01 之间的信息

```
select * from Notice where NcreateTime between '2012-12-01' and '2013-01-01'
```

（8）查询类型名为科研通知的通知公告信息

```
select * from Notice where Ntype = (select Tno from Type where TtypeName = '科研通知')
```

（9）查询类型名不是为科研通知的通知公告信息

```
select * from Notice where Ntype not in(select Tno from Type where TtypeName = '科研通知')
```

查询中常用到一些聚合函数，表 4-7 所示为常用的聚合函数。

表 4-7 聚合函数

函数名称	说　　明
avg	求平均值
count	求项数，返回 int 类型整数

续表

函数名称	说　　明
max	求最大值
min	求最小值
sum	返回表达式中所有值的和

(1) 查询类型为 1 的 notice 的个数

```
select count( * ) from Notice where Ntype = 1
```

(2) 查询类型表中 Tno 最大的信息

```
select max(Tno) from type
```

(3) 查询通知公告信息按类型分组的个数

```
select count( * ) from Notice group by Ntype
```

7. 删除表或是数据库语句 drop

语法形式如下:

```
drop table| database table_name| database_name.
```

4.3　SQL Server 2008 数据库管理系统

SQL Server 2008 数据库管理系统是一个可信任的、高效的、智能的数据平台,利用它可以有效地管理数据库。SQL Server 2008 的下载和安装在这里不介绍了。

1. 启动 SQL Server 2008

为了在 Web 应用程序访问 SQLSever 2008 管理的数据库,必须启动 SQL Server 2008 提供的数据库服务器。如果已经安装 SQL Server 2008,首先启动 SQL Server 服务。

启动服务有两种方式:

- 所有程序→Microsoft SQL Server 2008→配置工具→SQL Server 配置管理器→SQL Server 服务→SQL Server(MSSQLSERVER)启动
- 控制面板→服务→SQL Server(MSSQLSERVER)启动

2. 建立数据库

打开"Microsoft SQL Server Management Studio"。在“数据库”目录下是已有的数据库的名称,右键单击“数据库”可以建立新的数据库,我们新建立的数据库名称是" notice",如图 4-2 所示。

3. 创建表

在建立的数据库 notice 中,右键单击“表”可以建立新的表,创建名字为 Nuser 的表。表的属性(字段)为:Uno(编号),Uname(用户名)和 Upassword(密码),数据类型和主键参照表 4-4,如图 4-3 所示。

其他两个表类似地创建。通知公告系统的数据库和表建立完毕。

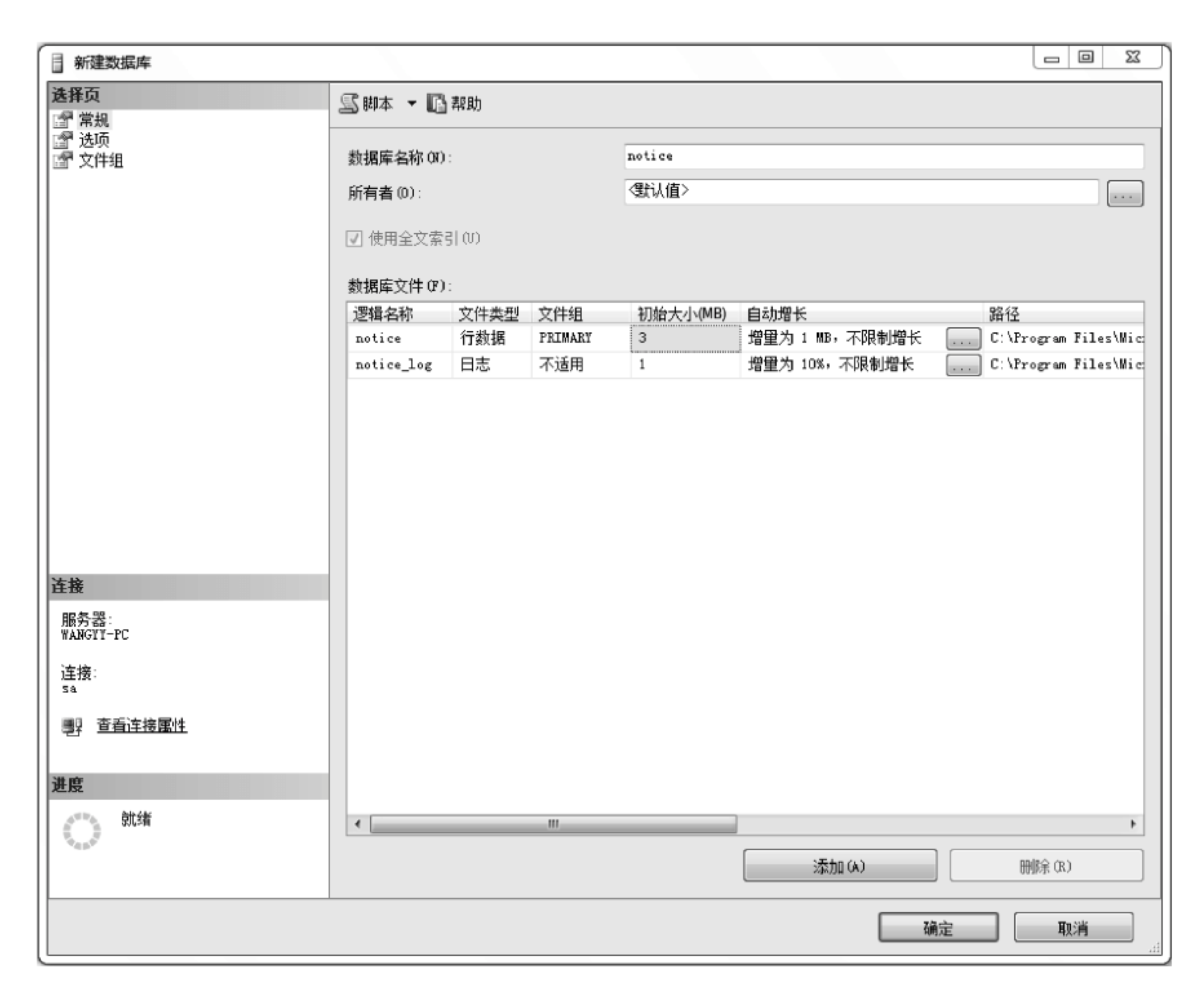


图 4-2 创建数据库 notice



图 4-3 创建表 Nuser

4.4 JDBC

我们都知道,Java 语言具有稳健、安全、易于使用、易于理解等特性,为开发数据库应用提供了良好的语言基础,而 JDBC 则充当了 Java 应用程序与各种不同数据库之间进行对话的媒介。JDBC 是 Java 数据库连接(Java DataBase Connection)技术的简称,它提供连接各种常用数据库的能力。有了 JDBC,访问各种数据库就是一件很容易的事。换言之,有了 JDBC,就不必为访问 Sybase 数据库专门写一个程序,为访问 Oracle 数据库又专门写一个程序,为访问 SQL Server 数据库又写另一个程序。我们只需要用 JDBC 写一个程序就够了。

4.4.1 JDBC 程序的工作原理

既然 JDBC 如此重要,我们就赶快来了解一下它的工作原理吧。JDBC 程序的工作原理如图 4-4 所示。

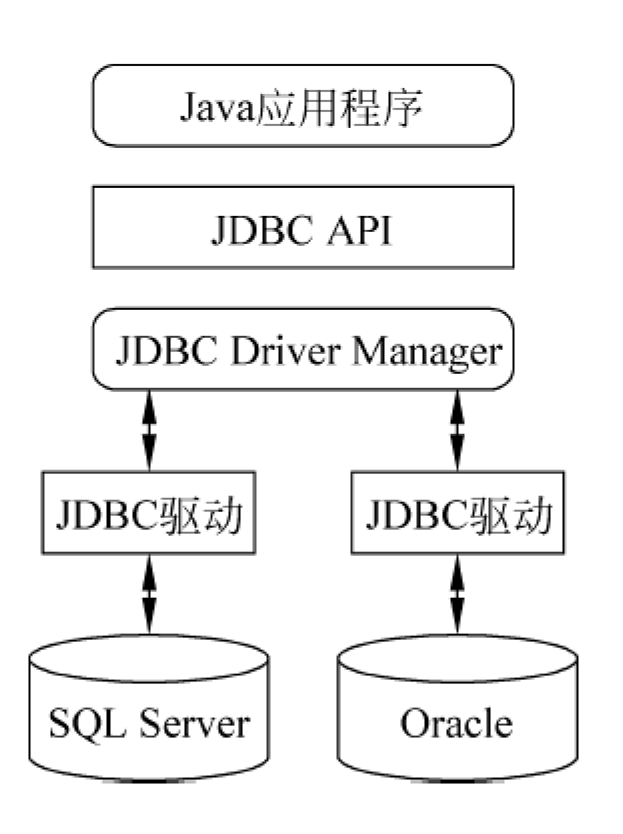


图 4-4 JDBC 工作原理

从图 4-4 中可以看到一个 JDBC 程序的几个重要的组成要素。最顶层当然是我们自己编写的 Java 应用程序,Java 应用程序可以使用集成在 JDK 中的 java.sql 和 javax.sql 包中的 Java API 来连接和操作数据库。下面我们就采用从上到下的顺序依次讲解剩余的几个要素。

1. JDBC API　JDBC API 由 Sun 公司提供,它提供了 Java 应用程序与各种不同数据库交互的标准接口,如:Connection(连接)接口、Statement 接口、PreparedStatement 接口、ResultSet(结果集)接口等。开发者使用这些 JDBC 接口进行各类数据库操作。

2. JDBC Driver Manager 由 Sun 公司提供,它能够管理各种不同的 JDBC 驱动。

3. JDBC 驱动　JDBC 驱动由各个数据库厂商提供,负责连接各种不同的数据库。如在图 4-4 中所示,微软公司为我们提供了专门连接 SQL Server 的 JDBC 驱动,而 Oracle 公司则为我们提供了专门连接 Oracle 的 JDBC 驱动。这些 JDBC 驱动实现了 JDBC API 中定义的各种接口。在开发 Java 应用程序时,我们只需正确加载 JDBC 驱动,正确调用 JDBC API 进行数据库的访问。

4.4.2 JDBC API

图 4-5 为我们展示了在使用 JDBC API 时,JDBC API 工作的 4 个重要环节:

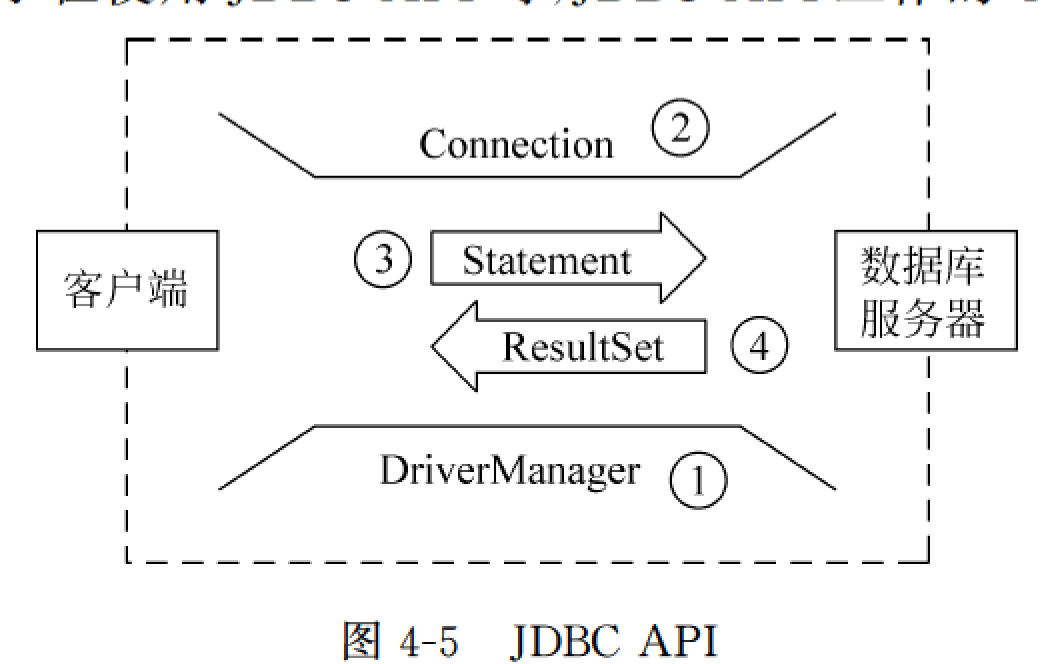


图 4-5 JDBC API

1. DriverManager 类：依据数据库的不同，管理 JDBC 驱动；
2. Connection 接口：负责连接数据库并担任传送数据的任务；
3. Statement 接口：由 Connection 产生，负责执行 SQL 语句；
4. ResultSet 接口：负责保存 Statement 执行后所产生的查询结果。

4.4.3 JDBC 程序的代码模板

利用 JDBC 实现数据库的操作一般可以分为如下几个步骤：

1. 加载 JDBC 驱动程序
2. 获取连接接口
3. 创建 Statement 对象
4. 执行 Statement 对象
5. 查看返回的结果集
6. 关闭结果集对象
7. 关闭 Statement 对象
8. 关闭连接接口

下面就具体的解释一下这几个步骤。

第一步：把 JDBC 驱动类装载入 Java 虚拟机中。为此，可使用 Class.forName()方法，此方法将给定的类加载到 Java 虚拟机中。如果系统中不存在给定的类，则会引发异常，异常类型为 ClassNotFoundException。代码示例：

```
Class.forName("JDBC 驱动类的名称");
```

第二步：加载驱动，并与数据库建立连接。DriverManager 类跟踪已注册的驱动程序，当调用 getConnection()方法时，它会搜索整个驱动程序列表，直到找到一个能够连接至数据连接字符串中指定的数据库的驱动程序。加载此驱动程序之后，将使用 DriverManager 类的 getConnection()方法建立与数据库的连接。此方法接收三个参数，分别表示 URL、用户名和密码。用户名和密码是可选的。代码示例：

```
Connection con = DriverManager.getConnection(数据库连接字符串,数据库用户名,密码);
```

第三、四步：发送 SQL 语句，并得到结果集。一旦连接建立，就使用该连接创建 Statement 接口的实例，并将 SQL 语句传递给它所连接的数据库，并返回类型为 ResultSet 的对象，它包含执行 SQL 查询的结果。代码示例：

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table");
```

第五步：处理结果。使用 ResultSet 对象的 next()方法将光标(cursor)指向下一行。最初光标位于第一行之前，因此第一次调用 next()方法将把光标置于第一行上。如果到达结果集的末尾，则 ResultSet 的 next()方法会返回 false。方法 getXXX 提供了获取当前行中某列值的途径，列名或列号可用于标识要从中获取数据的列。例如，如果数据表中第一列的列名为 a，存储类型为整型，则可以使用两种方法获取存储在该列中的值，例如：int x=rs.getInt("a")；或者：int x = rs.getInt(1)；处理结果的代码示例：

```
while(rs.next){
    int x = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
}
```

第六、七、八步为关闭对象，释放有效资源

示例4－1

```
/**
 * JDBC 程序的代码模板
 */
try {
      Class.forName(JDBC 驱动类);
} catch (ClassNotFoundException e) {
      System.out.println("无法找到驱动类");
}
try {
      Connection con = DriverManager.getConnection(JDBC URL,数据库用户名,密码);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
      while (rs.next()) {
            int x = rs.getInt("a");
            String s = rs.getString("b");
            float f = rs.getFloat("c");
      }
} catch (SQLException e) {
      e.printStackTrace();
}finally{
    rs.close();
    stmt.clse();
    con.close();
}
```

问答

问题：什么是 JDBC URL?

答：JDBC URL 提供了一种标识数据库的方法，可以使相应的 JDBC 驱动程序能识别数据库并与之建立连接。实际上，在编写 Java 应用程序时，我们不必关心如何来形成 JDBC URL，只需使用与 JDBC 驱动程序一起提供的 URL 即可。

JDBC URL 的标准语法由以下三个部分组成，各部分间用冒号分隔。

jdbc:<子协议>:<子名称>

JDBC URL 的三个部分可以分解如下：jdbc——代表协议；<子协议>——驱动程序名或数据库连接机制的名称；<子名称>——一种标识数据库的方法。

以下为两个比较典型的 JDBC URL 示例，将在稍后详细加以介绍。

(1) jdbc:odbc:notice

（2）jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=notice

比较

Statement 与 PreparedStatement 的区别

我们知道获取 Connection 对象之后，就可以使用 Connection 对象生成的 Statement 实例，进行数据库的操作：executeQuery（String sql），执行 SQL 查询获取对象；executeUpdate(String sql)，执行插入、删除、更新等操作，返回影响的行数；excute(String sql)最一般的执行方法。

PreparedStatement 接口继承自 Statement 接口，PreparedStatement 比 Statement 对象使用起来更加的灵活。PreparedStatement 实例包含已编译的 SQL 语句，SQL 语句可有一个或多个输入参数，用“?”作为占位符。由于 PreparedStatement 对象已经预编译，所以执行速度快于 Statement。

4.4.4 JDBC 驱动

JDBC 驱动由数据库厂商提供，在我们实际编程过程中，有两种较为常用的驱动方式。第一种是 JDBC-ODBC 桥连，适用于个人开发与测试，它通过 ODBC 与数据库进行连接，另一种是纯 Java 驱动，它直接同数据库连接，在生产型开发中推荐使用纯 Java 驱动方式。这两种连接方式的示意图如图 4-6 所示。

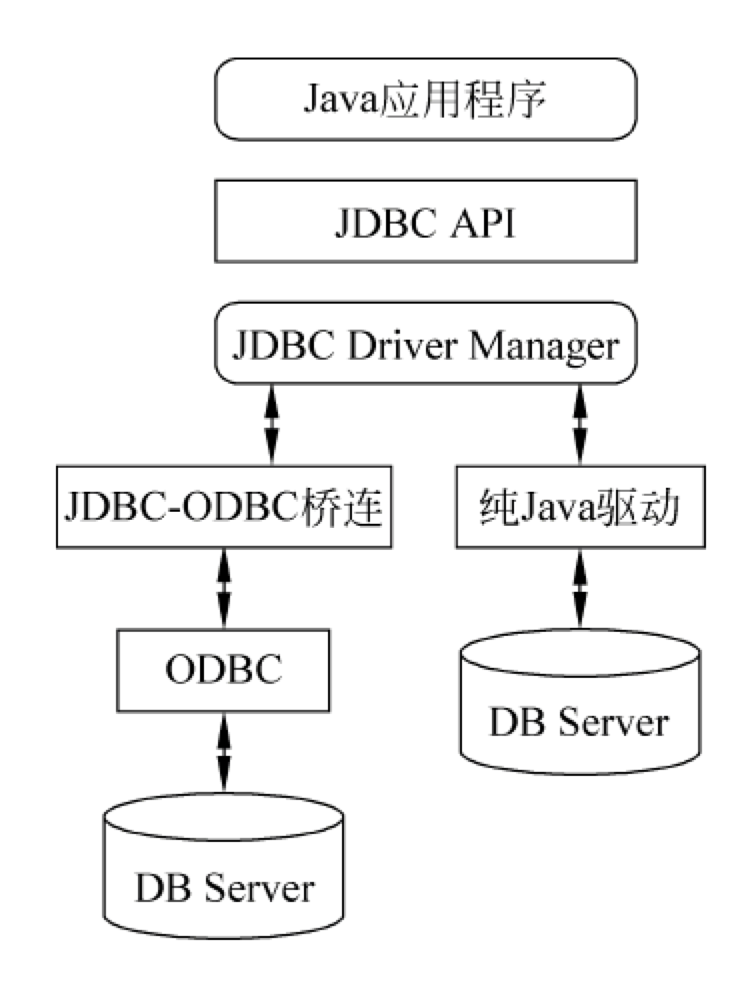


图 4-6 两种常用的驱动方式

1. JDBC-ODBC 桥连　它将对 JDBC API 的调用转换为对另一组数据库连接（即 ODBC）API 的调用。如图 4-7 所示。

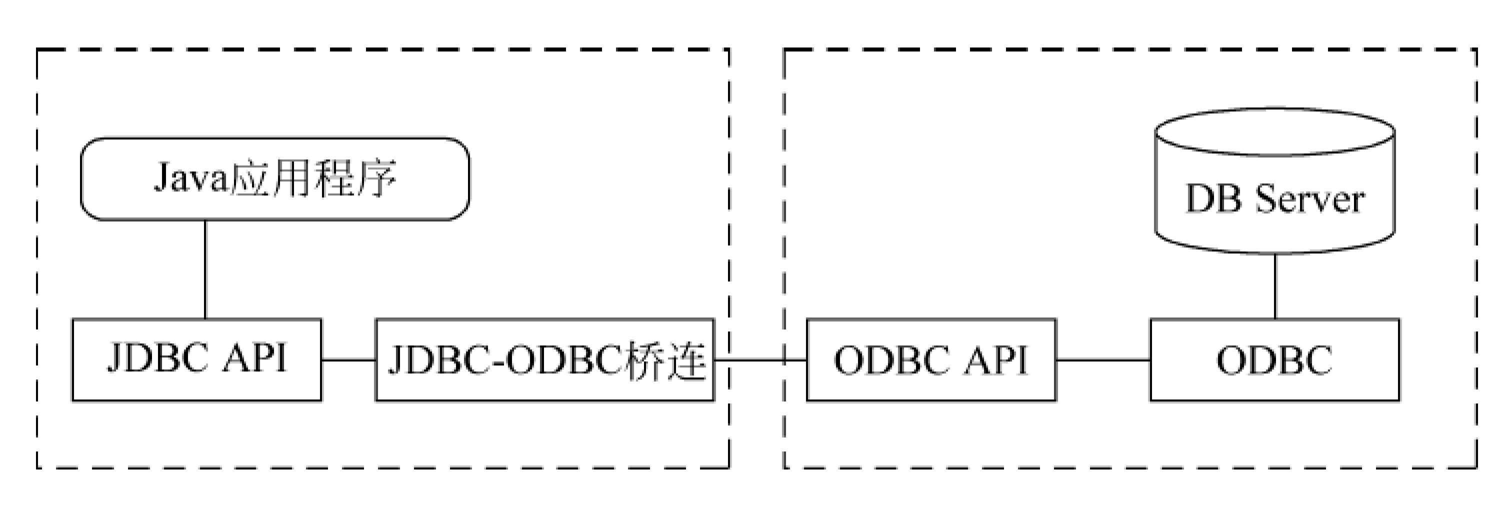


图 4-7 JDBC-ODBC 桥连

JDK 中已经包括了 JDBC-ODBC 桥连的驱动接口，所以进行 JDBC-ODBC 桥连时，不需要额外下载 JDBC 驱动程序，只需要配置 ODBC 数据源即可，具体配置步骤如下：

(1) 选择"开始"→"控制面板"→"管理工具"→"数据源(ODBC)"选项，打开"ODBC 数据源管理器"对话框，如图 4-8 所示。



图 4-8 "ODBC 数据源管理器"对话框

(2) 选择"系统 DSN"选项卡，单击"添加"按钮，打开"创建新数据源"对话框，如图 4-9 所示。



图 4-9 "创建新数据源"对话框

(3) 在对话框中选中"SQL Server"选项，然后单击"完成"按钮，打开"创建到 SQL Server 的新数据源"对话框，如图 4-10 所示。

在名称一栏添入"notice"，服务器一栏添入"localhost"，单击"下一步"按钮。

(4) 在"DSN 配置"中选择使用用户输入登录 ID 和密码的 SQL Server 验证。登录 ID：sa，密码：123456。如图 4-11 所示，单击"下一步"按钮。

(5) 在"更改默认的数据库为"中，选择已经建好的 notice 数据库，单击"下一步"按钮，如图 4-12 所示。



图 4-10 “创建到 SQL Server 的新数据源”对话框



图 4-11 DSN 配置



图 4-12 更改默认的数据库为 notice



(6)单击“完成”按钮后,结果如图 4-13 所示。



图 4-13　配置 ODBC

(7)单击“测试数据源”按钮,出现如图 4-14 所示的对话框。

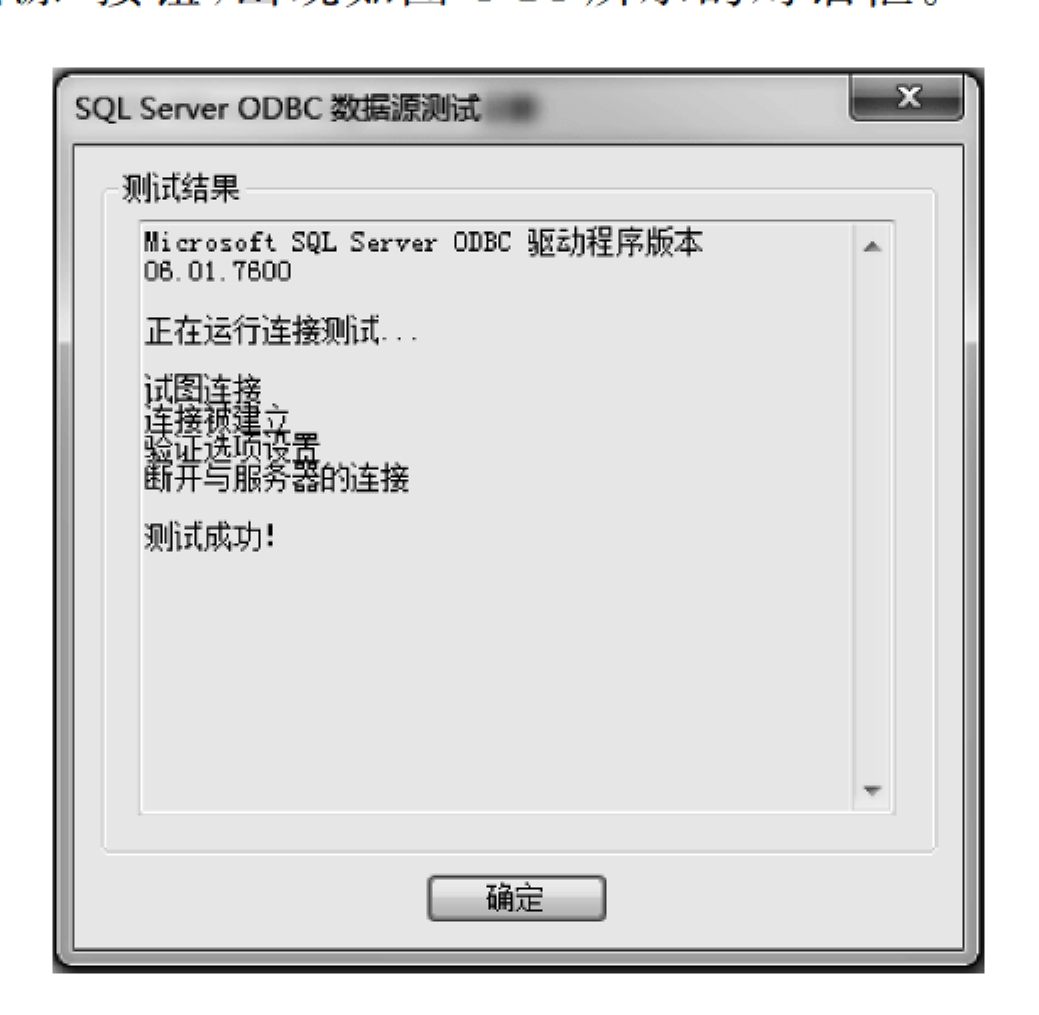


图 4-14　测试数据源成功

单击“确定”按钮,配置成功。

需要注意的是:虽然通过 JDBC-ODBC 桥连的方式可以访问所有 ODBC 可以访问的数据库,但是 JDBC-ODBC 桥连不能提供非常好的性能,一般不适合在实际系统中使用。

如果我们使用 JDBC-ODBC 桥进行数据库连接,那么 JDBC 驱动类的名称为

```
sun.jdbc.odbc.JdbcOdbcDriver,
```

数据库连接字符串将以 jdbc:odbc:开始,后面跟随数据源名称。因此,假设我们已经配置了一个叫 notice 的 ODBC 数据源,数据库连接字符串就是 jdbc: odbc:notice,假定登录数据库系统的用户名为 sa,口令为 123456,只需下面的两行代码就可以建立一个数据库连接:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;
Connection con = DriverManager. getConnection (" jdbc: odbc: notice","sa","123456");
```

示例4-2

```
/**
 * JDBC-ODBC 连接数据库测试
 */
package com.bean;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
public class JDBCODBCTest {
    public static void main(String[ ] args) {
        Connection dbConnection = null;
        PreparedStatement pStatement = null;
        ResultSet res = null;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;
               dbConnection = DriverManager.getConnection ("jdbc:odbc:notice","sa",
"123456");
            String strSql = "select * from Notice ";
            pStatement = dbConnection.prepareStatement(strSql);
            res = pStatement.executeQuery();
            while (res.next()) {
                String title = res.getString("Ntitle");
                System.out.println("标题" + title);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try{
                if(res != null)res.close();
                if(pStatement != null)pStatement.close();
                if(dbConnection != null)dbConnection.close();
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

运行控制台输出结果，说明 JDBC-ODBC 桥连成功。

2. 纯 Java 驱动方式 它由 JDBC 驱动直接访问数据库，驱动程序完全由 Java 语言编写，运行速度快，而且具备了跨平台的特点。但是，由于这类 JDBC 驱动是数据库厂商特定的，即这类 JDBC 驱动只对应一种数据库，因此访问不同的数据库需要下载专用的 JDBC 驱动。如图 4-15 所示，描述了纯 Java 驱动方式的工作原理。

如果我们使用纯 Java 驱动方式进行数据库连接，首先需要下载数据库厂商提供的驱程序 Jar 包，并将 Jar 包引入工程中。本书使用的数据库是 SQL Server 2008，因此，可以从微

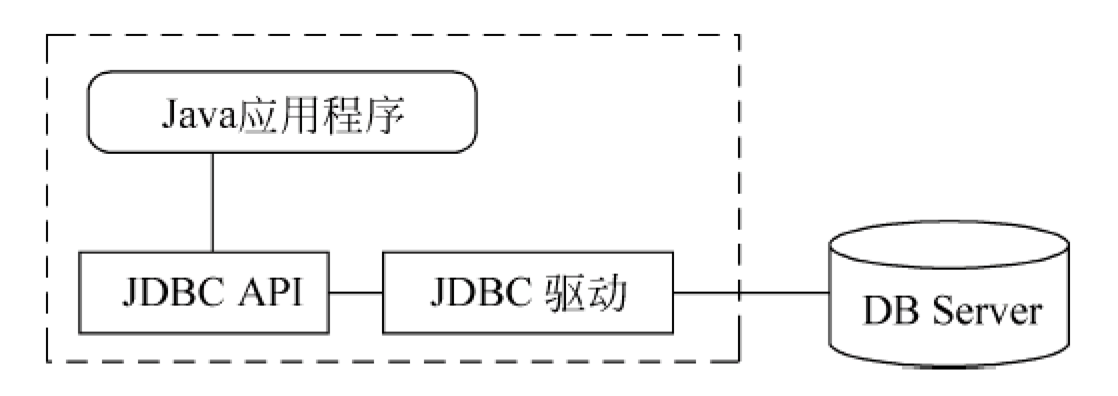


图 4-15 纯 Java 驱动方式

软的官方网站下载驱动程序 Jar 包，并查看相关帮助文档，获得驱动的名称以及数据库连接字符串。接下来，就可以进行编程，与数据库建立连接。代码示例：

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
Connection con =
DriverManager.getConnection("jdbc:sqlserver://localhost:1433;DatabaseName = notice","sa", "
123456");
```

示例4-3

```
/**
 * JDBC 连接数据库测试类
 */
package com.bean;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
public class JDBCTest {
    public static void main(String[ ] args) {
        Connection dbConnection = null;
        PreparedStatement pStatement = null;
        ResultSet res = null;
        try {
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            dbConnection = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;
DatabaseName = notice","sa", "123456");
            String strSql = "select * from Notice ";
            pStatement = dbConnection.prepareStatement(strSql);
            res = pStatement.executeQuery();
            while (res.next()) {
                String title = res.getString("Ntitle");
                System.out.println("标题：" + title);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try{
                if(res != null)res.close();
```

```
                    if(pStatement != null)pStatement.close();
                    if(dbConnection != null)dbConnection.close();
                }catch(Exception e){
                    e.printStackTrace();
                }
            }
        }
    }
```

示例 4-3 在控制台输出效果如图 4-16 所示。

```
Problems  Tasks  Web Browser  Console
<terminated> JDBCTest [Java Application] D:\Program Files\Ja
标题：1课表查询通知
标题：2放假通知
标题：3考试通知
标题：4招聘通知
标题：6过年放假通知
标题：5下雪了
```

图 4-16　JDBC 连接数据库测试控制台显示效果

4.5　贯穿项目 JDBC 应用

在贯穿项目通知公告发布系统中，我们采用纯 Java 驱动方式连接数据库。又由于项目的多个类都需要连接数据库，因此创建一个工具类用于连接数据库。在 com.dao 中创建工具类 ConnectionManager.java。

示例4-4

```
/**
 * ConnectionManager.java
 */
package com.dao;

import java.sql.*;

public class ConnectionManager {
     private static final String DRIVER_CLASS = "com.microsoft.sqlserver.jdbc.
SQLServerDriver";
     private static final String DATABASE_URL = "jdbc:sqlserver://localhost:1433;
DatabaseName=notice";
    private static final String DATABASE_USRE = "sa";
    private static final String DATABASE_PASSWORD = "123456";
    /**
     * 返回连接
     */
    public static Connection getConnection() {
```

```
        Connection dbConnection = null;
        try {
            Class.forName(DRIVER_CLASS);
            dbConnection = DriverManager.getConnection(DATABASE_URL,
                        DATABASE_USRE, DATABASE_PASSWORD);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return dbConnection;
    }
    /**
     * 关闭连接
     */
    public static void closeConnection(Connection dbConnection) {
        try {
            if (dbConnection != null && (!dbConnection.isClosed())) {
                dbConnection.close();
            }
        } catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
        }
    }
    /**
     * 关闭结果集
     */
    public static void closeResultSet(ResultSet res) {
        try {
            if (res != null) {
                res.close();
                res = null;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    /**
     *关闭语句
     */
    public static void closeStatement(PreparedStatement pStatement) {
        try {
            if (pStatement != null) {
                pStatement.close();
                pStatement = null;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```



编写测试类连接数据库，利用工具类 ConnectonManager.java。

```
    /**
     * 利用工具类 测试连接数据库
     */
    package com.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class TestConnection {

    public static void main(String[ ] args) {
        Connection dbConnection = null;
        PreparedStatement pStatement = null;
        ResultSet res = null;
        try {
            dbConnection = ConnectionManager.getConnection();
            String strSql = "select * from Nuser ";
            pStatement = dbConnection.prepareStatement(strSql);
            res = pStatement.executeQuery();
            while (res.next()) {
                System.out.println(res.getString("Uname"));
            }
        } catch (SQLException sqlE) {
            sqlE.printStackTrace();
        } finally {
            ConnectionManager.closeResultSet(res);
            ConnectionManager.closeStatement(pStatement);
            ConnectionManager.closeConnection(dbConnection);
        }
    }
}
```

4.6 上机练习

1. 在 SQL Server2008 中创建通知公告发布系统所需的数据库、表。

需求说明：创建数据库 notice，创建表 Notice、Nuser 和 Type，并插入测试数据。

实现思路：参考 4.3 节。

2. 执行 SQL 语句。

需求说明：练习 SQL 语句 select、insert、update、delete、drop、create 的用法。

实现思路：参考 4.3 节。

3. 配置 ODBC 数据源。

需求说明：利用 JDBC-ODBC 桥连配置 ODBC 数据源，并编写测试类，测试连接是否成功。

实现思路：参考 4.4.4 节。

4. 使用纯 Java 驱动方式进行数据库连接。

需求说明：使用纯 Java 驱动方式进行数据库连接，并编写测试类，测试连接是否成功。

实现思路：参考 4.4.4 节。

5. 编写工具类。

需求说明：为通知公告发布系统，编写连接数据库的工具类，并编写测试类，测试连接是否成功。

实现思路：参考 4.5 节。

4.7 总 结

(1) 数据库(DB-DataBase)是指在计算机内按一定形式存放、有组织、统一管理的相关数据和数据库对象的集合。

(2) SQL (Structured Query Language)是结构化查询语言的简称。

(3) SQL 有以下 3 部分组成：

- 数据定义语言(Data Definition Language，DDL)
- 数据操纵语言(Data Manipulation Language，DML)
- 数据控制语言(Data Control Language，DCL)

(4) 常用的标准 SQL 命令，如 select、insert、update、delete、drop、create 等，几乎可以完成所有的数据库的操作。

(5) JDBC 是 Java 数据库连接(Java DataBase Connection)技术的简称，提供连接各种常用数据库的能力。

(6) JDBC API 工作的 4 个重要环节：DriverManager 类(依据数据库的不同，管理 JDBC 驱动)；Connection 接口(负责连接数据库并担任传送数据的任务)；Statement 接口(由 Connection 产生，负责执行 SQL 语句)；ResultSet 接口(负责保存 Statement 执行后所产生的查询结果)。

(7) 利用 JDBC 实现数据库的操作一般可以分为如下几个步骤：

- 加载 JDBC 驱动程序
- 获取连接接口
- 创建 Statement 对象
- 执行 Statement 对象
- 查看返回的结果集
- 关闭结果集对象
- 关闭 Statement 对象
- 关闭连接接口

(8) JDBC 驱动由数据库厂商提供，在实际编程过程中，有两种较为常用的驱动方式。第一种是 JDBC-ODBC 桥连，适用于个人开发与测试，它通过 ODBC 与数据库进行连接；另一种是纯 Java 驱动，它直接同数据库连接，在生产型开发中推荐使用纯 Java 驱动方式。

4.8 作　　业

一、选择题

1. 在 JSP 中使用 JDBC 语句访问数据库,正确导入 SQL 类库的语句是(　　)。

A. <%@page import="java.sql.*"%>　　B. <%@page import=" sql.*"%>

C. <% page import="java.sql.*"%>　　D. <%@ import="java.sql.*"%>

2. 在 JDBC API 中提供的(　　)类的作用是:依据数据库的不同,管理不同的 JDBC 驱动程序。

A. DriverManager　　B. Connection　　C. Statement　　D. Class

3. 假设已经获得 ResultSet 对象 res,那么获取第一行数据的正确语句是(　　)。

A. res.hasNext()　　B. res.next()　　C. res.nextRow()　　D. res.hasNextRow()

二、简答题

1. 常用的 SQL 命令有哪些?

2. 利用 JDBC 实现数据库的操作的步骤是什么?

3. 简述 JDBC-ODBC 桥连的步骤。

三、程序题

1. 在 SQL Server 2008 中创建数据库 school,创建表 student,包含以下信息:学号、姓名、性别、年龄、联系电话、班级。

2. 查询 student 表中的年龄大于 20 的学生信息。

3. 删除 student 表中男生的信息。

4. 查询 student 表中班级名有“信科”两个字的学生信息。

5. 按照性别分组查询所有学生的平均年龄。

6. 创建项目,利用 JDBC 编写 Java 测试类,在控制台输出所有学生的姓名。

第 5 章　JSP 中的 JavaBean

本章学习目标

- 掌握使用 JavaBean 封装业务逻辑和数据
- 掌握如何在 JSP 中使用 JavaBean
- 掌握 JSP 的动作元素<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>

这一章我们来讲解 JSP 中的 JavaBean,那么 JSP 中为什么需要 JavaBean 呢?

5.1　为什么需要 JavaBean

J2EE 程序是基于组件开发的,就好像是玩具积木一样,从表面看各个积木块是一堆毫不相干的小木块,但是经过设计和安排,就可以把这些木块组装成我们想要的“建筑”。在程序中也同样存在类似的道理,Java 是一种面向对象的编程语言,在设计和解决问题时,都是以面向对象的思想进行的。例如,数据库连接类,在这个类中定义了连接方法和关闭方法,对于这个类来说,它的使命就是建立连接和关闭连接,是程序的一个组成部分,就好像一块积木在整个积木作品中的作用一样。一个积木作品是由很多个相同或者不同的积木块组成,而程序同样也是这样。

在之前的开发中,我们在 JSP 页面中嵌入了大量的 Java 代码,这些代码负责完成业务逻辑、数据显示等操作。JSP 页面中大量的 Java 代码与 HTML 标签混杂,导致了维护和修改上的困难。为了分离页面中的 HTML 代码和 Java 代码,一个非常简单的想法就是编写一些类来封装页面的数据和业务逻辑。这样只需几行代码调用类中的方法,就可以实现所需的功能。采用这种方式不但能够使代码重用,还能使数据显示和业务逻辑分开,在 JSP 技术中,可以使用 JavaBean 组件来实现。

5.2　什么是 JavaBean

JavaBean 是 Java 开发中可以跨平台的重用组件。JavaBean 在 JSP 程序中常用来封装业务逻辑、数据库操作等。JavaBean 组件本质上就是一个 Java 类,只不过这个类需要遵循一些编码的约定,这个类可以重用。从 JavaBean 的功能上可以分为以下两类:

- 封装数据
- 封装业务

JavaBean 一般情况下要满足以下要求：

- JavaBean 是一个公有类，并提供无参的公有的构造方法
- 属性私有
- 具有公有的访问属性的 getter 和 setter 方法

符合上述条件的类，我们都可以把它看成是 JavaBean 组件。

在程序中，开发人员所要处理的无非是业务逻辑和数据，而这两种操作都可以使用 JavaBean 组件。一个应用程序中会使用很多 JavaBean。由此可见 JavaBean 组件是应用程序的重要组成部分。

5.3 封装数据

先来看一个简单的 JavaBean，在 MyEclipse 中要创建 JavaBean，首先在项目 src 中创建 com. bean 包，在包内创建 User 类，类的属性如示例 5-1。

示例5－1

```
/**
 * User 类
 */
package com.bean;
public class User {
    private int id;               //ID
    private String name;          //用户名
    private String password;      //密码
    /*
     * 不带参数的构造方法
     */
    public User() {}
}
```

对属性分别给与 setter 和 getter 方法，MyEclipse 提供了一个非常方便的快捷方法生成 setter 和 getter 方法，如图 5-1 及图 5-2 所示。

添加完 setter 和 getter 方法后的代码如示例 5-2。

示例5－2

```
/**
* User 类
 */
package com.bean;
public class User {
    private int id;               //ID
    private String name;          //用户名
```

```
        private String password;        //密码
        /*
         * 构造方法
         */
        public User() {}

        public int getId() {
            return id;
        }
        public void setId(int id) {
            this.id = id;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public String getPassword() {
            return password;
        }
        public void setPassword(String password) {
            this.password = password;
        }
    }
```

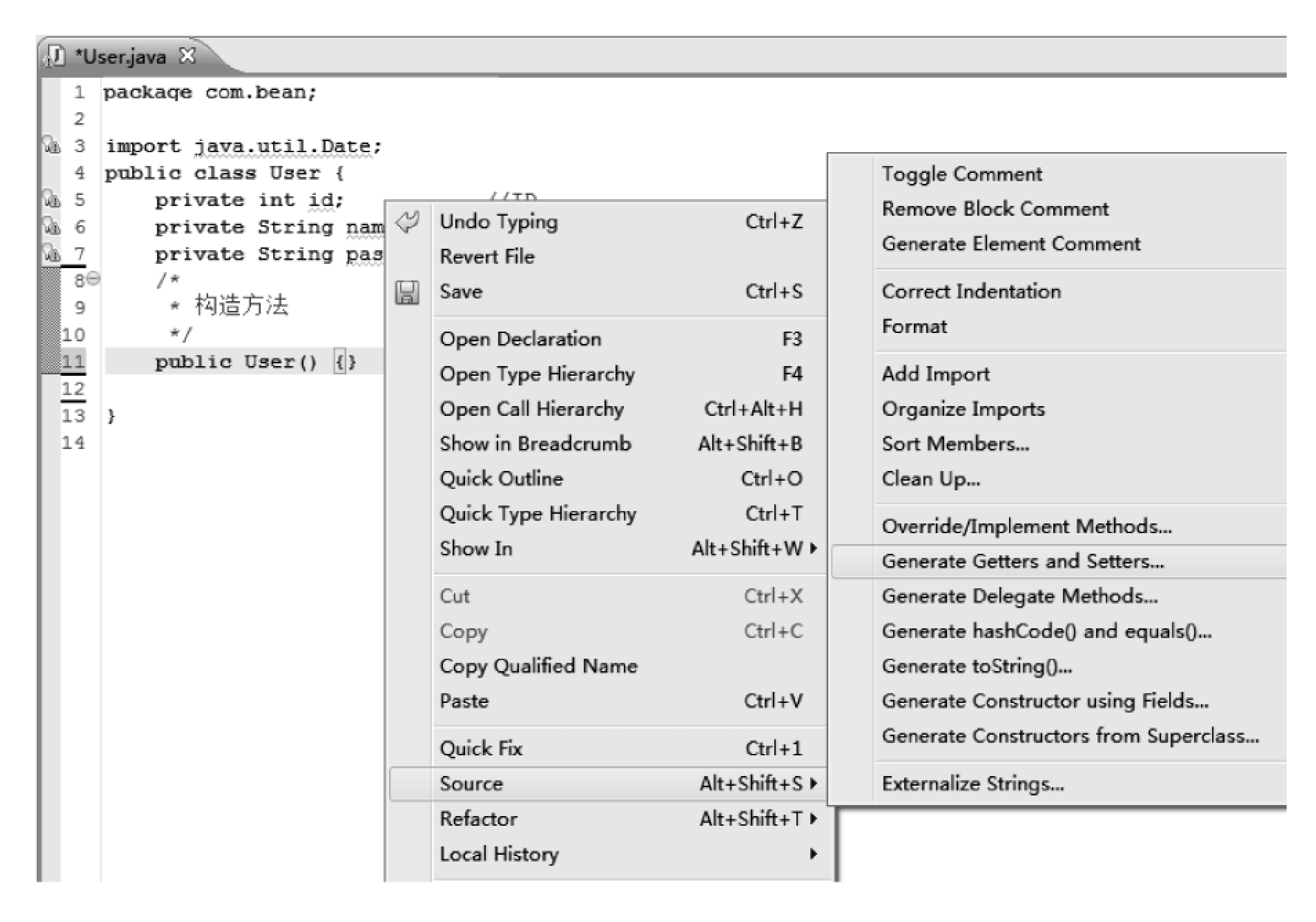


图 5-1 MyEclipse 中添加 setter 和 getter 方法(一)

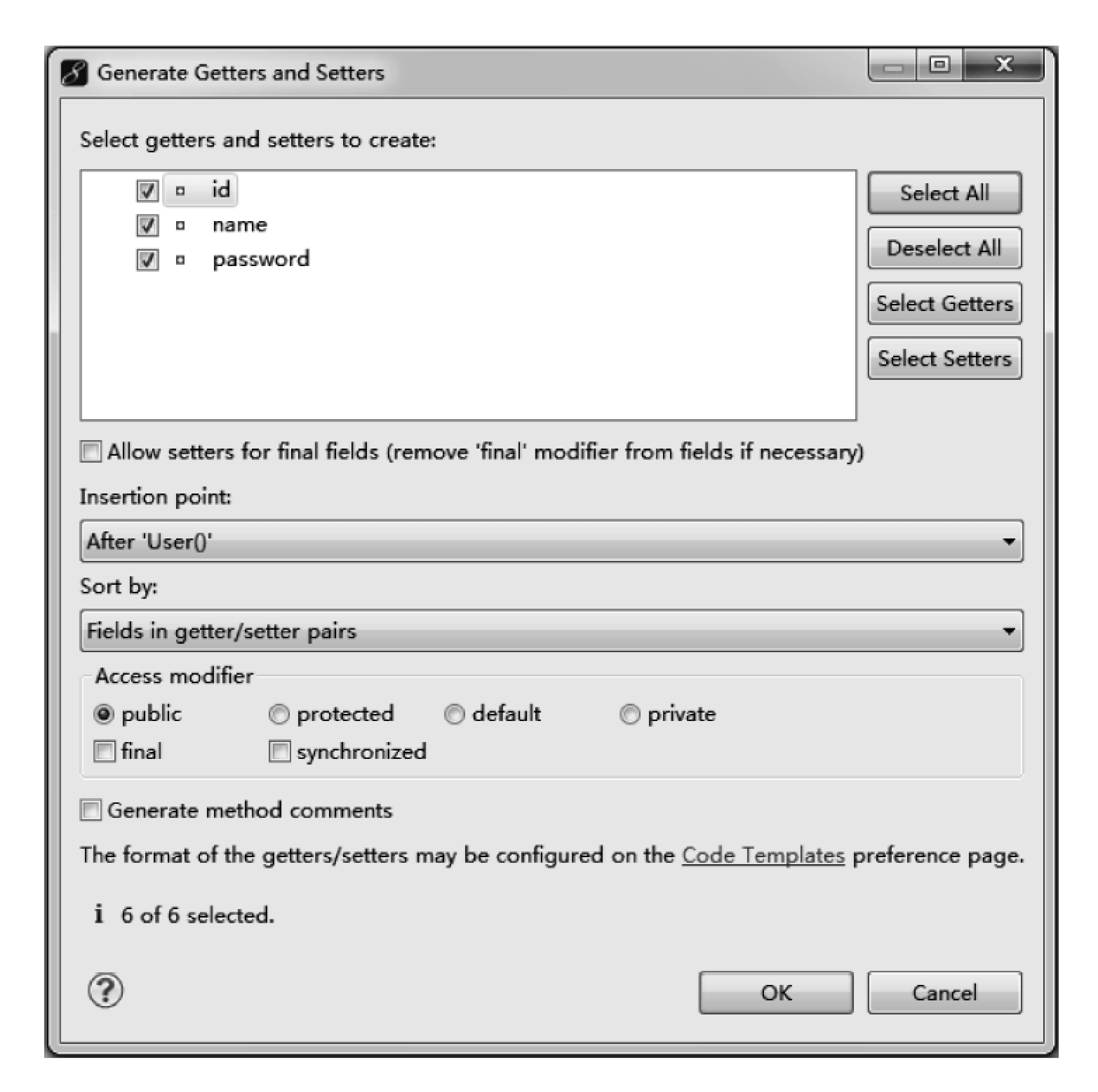


图 5-2 MyEclipse 中添加 setter 和 getter 方法(二)

这是一个很典型的封装数据 JavaBean。这个 JavaBean 封装了通知公告发布系统用户表的数据，id、name、password 等是用户的属性，外部通过 getter / setter 可以对这些属性进行操作。



资料

Sun 推荐的属性命名规则为：xxx 的属性对应 setXxx()方法。

一般情况下，Java 的属性变量都以小写字母起头，如 name、userName 等。但也存在特殊的情况，如一些特定意义的大写英文缩写(如 XML，URL 等)。JavaBean 也允许大写字母起头的属性变量名，不过必须满足“变量的前两个字母要么全部大写，要么全部小写”的要求，如 IDCode、ID、ICCard 等属性变量名是合法的，而 iD、iCcard、iDCode 等属性变量名则是非法的。在有些情况下非法的变量名再以属性名第一个字母大写的方式命名 get 和 set 方法，会导致找不到属性错误。

解决办法：

- 如果属性名的第二个字母大写，那么该属性名直接用作 getter/setter 方法中 get/set 的后部分，就是说大小写不变。例如属性名为 uName，方法是 getuName/setuName。
- 如果前两个字母大写(一般的专有名词和缩略词都会大写)，则属性名直接用作 getter/setter 方法中 get/set 的后部分。例如属性名为 URL，方法是 getURL/setURL。
- 如果首字母大写，则属性名直接用作 getter/setter 方法中 get/set 的后部分。例如

属性名为 Name，方法是 getName/setName，这是最糟糕的情况，会找不到属性出错的地方，因为默认的属性名是 name。

5.4 封装业务

在编写程序的时候，一个封装数据的 JavaBean 一般情况下对应着数据库内的一张表（或视图），JavaBean 的属性与表（或视图）内字段的属性一一对应。同样，相对于一个封装数据的 JavaBean 一般都会有一个封装该类的业务逻辑和业务操作的 JavaBean 相对应。

例如在通知公告发布系统中，浏览页面可以显示所有的通知公告类型，通知公告类型保存在数据库的 Type 表中，类型对应的数据 JavaBean 是 Type.java，那么封装数据 Type 对应的业务逻辑 JavaBean 是 TypeControl.java 代码如示例 5-3。

示例5-3

```
/**
 * Type 业务处理 JavaBean 类
 */
package com.bean;
import com.bean.Type; …
public class TypeControl {
    public List getAllTypeList() {
        ArrayList list = new ArrayList();
        Connection dbConnection = null;
        PreparedStatement pStatement = null;
        ResultSet res = null;
        try {
      Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
     dbConnection = DriverManager. getConnection ( " jdbc: sqlserver://localhost: 1433;
DatabaseName=notice","sa", "123456");
            String strSql = "select * from Type ";
            pStatement = dbConnection.prepareStatement(strSql);
            res = pStatement.executeQuery();
            while (res.next()) {
                int id = res.getInt("Tno");
                String typeName = res.getString("TtypeName");
                //把各属性封装到一个 type 对象中
                Type type = new Type(id, typeName);
                //把各 type 对象添加到集合 list 中
                list.add(type);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try{
                if(res != null)res.close();
                if(pStatement != null)pStatement.close();
```

```
                    if(dbConnection != null)dbConnection.close();
                    }catch(Exception e){
                    e.printStackTrace();
                }
            }
            return list;
        }
    }
```

在第 4 章中我们已经编写了连接数据库的工具类(示例 4-4)ConnectionManager.java，因此 TypeControl.java 可以简化为示例 5-4 的形式。

示例5－4

```
/**
 * Type 业务处理 JavaBean 类
 */
package com.dao;
import java.sql.*;
import java.util.*;
import com.bean.Type;
public class TypeControl {
    public List getAllTypeList() {
        ArrayList list = new ArrayList();
        Connection dbConnection = null;
        PreparedStatement pStatement = null;
        ResultSet res = null;
        try {
            dbConnection = ConnectionManager.getConnection();
            String strSql = "select * from Type ";
            pStatement = dbConnection.prepareStatement(strSql);
            res = pStatement.executeQuery();
            while (res.next()) {
                Type type = new Type();
                type.setId(res.getInt("Tno"));
                type.setTypeName(res.getString("TtypeName"))
                list.add(type);
            }
        } catch (SQLException sqlE) {
            sqlE.printStackTrace();
        } finally {
            ConnectionManager.closeResultSet(res);
            ConnectionManager.closeStatement(pStatement);
            ConnectionManager.closeConnection(dbConnection);
        }
        return list;
    }
}
```

5.5 JSP 与 JavaBean

现在我们已经掌握了如何创建封装数据的 JavaBean 和封装业务逻辑的 JavaBean，但是在 JSP 中如何使用 JavaBean 呢？在 JSP 页面中，可以像使用普通类一样，实例化一个 JavaBean 对象，然后根据需要调用该对象的方法。在 JSP 中引入并使用 JavaBean 的语法如下。

```
//引入 JavaBean
<%@ page import="com.bean.Notice" %>
//使用 JavaBean
<%
    Notice notice = new Notice();
    notice.setId(15);
    notice.setTitle("Hello");
%>
```

在 JSP 中使用 JavaBean 就像我们在 Java 程序中编写类是一样的，实例化 JavaBean 后，就可以使用其中的方法了。

5.6 JSP 动作元素

第 3 章中我们已经学习的动作元素包括<jsp:include>、<jsp:plugin>、<jsp:forward>、<jsp:param>等。本节介绍另外三个和 JavaBean 结合非常的紧密动作元素：<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>。

5.6.1 <jsp:useBean>

<jsp:useBean>动作元素可以在 JSP 页面中创建一个 Bean 实例，并且可以通过属性的设置将该实例储存到 JSP 指定范围内。此动作元素的语法如下。

```
<jsp:useBean
    id="beanInstanceName"
    scope="page|request|session|application"
    class="package.class"
></jsp:useBean>
```

其中，id 指定该 JavaBean 的实例变量的名称，scope 指定该 Bean 变量的有效范围。

page 指只在当前 JSP 页面中有效，request 指在任何执行相同请求的 JSP 文件中使用 Bean，直到页面执行完毕，session 指从创建该 Bean 开始，在相同 session 下的 JSP 页面中可以使用该 Bean，application 指从创建该 Bean 开始，在相同 application 下的 JSP 页面中可以使用该 Bean。

5.6.2 <jsp:setProperty>

<jsp: setProperty>动作元素通常与<jsp:useBean>一起使用，它调用 Bean 的 setXxx 方法，将请求的参数赋值给由<jsp:useBean>创建的 Bean 中对应的属性。该动作元素的语法如下。

```
<jsp:setProperty
    name = "beanInstanceName"
    property = " * " |
    property = "propertyName" |
    property = "propertyName" [param = "propertyName"] |
    property = " propertyName" value = " {string |<% = expression %>}"
/>
```

说明：其中 name 属性是用来指定一个存在 JSP 中某个范围内的 JavaBean 实例。<jsp: setProperty>将会按照 page、request、session、application 的顺序来查找 Bean 的实例，直到第一个被找到，如果在查找范围内不存在这个 Bean 实例，则出现异常信息。

property 的值为" * "，则 request 请求中所有参数的值将被赋给 Bean 中与参数具有相同名字的属性。如果请求中存在参数值为空，则对应的 Bean 属性将不会被设定(也不会设为 null)；如果 Bean 中存在一个属性，但请求中没有与之对应的参数，那么该属性同样不会被设定(也不会赋值为 null)，这两种情况下的 Bean 属性都会保留原来或默认的值。

```
property = "propertyName"
```

property 属性取值为 Bean 中的属性时，则只会将 request 请求中与该 Bean 属性同名的一个参数的值赋给这个 Bean 属性。例如，如果 property 属性指定的 Bean 属性为 userName，那么指定 Bean 中必须存在 setUserName()方法，否则会出现异常信息。

```
property = "propertyName" [param = "propertyName"]
```

property 属性指定一个 request 请求中的参数，property 属性指定 Bean 中的某个属性。该种使用方法允许将请求中的参数赋值给 Bean 中与该参数不同名的属性。如果 param 属性指定参数的值为空，那么由 property 属性指定的 Bean 属性会保留原来或默认的值而不会被赋为 null。

```
property = " propertyName" value = " {string |<% = expression %>}"
```

其中 value 属性指定的值可以是一个字符串数值或表示一个具体值的 JSP 表达式或 EL 表达式。该值将被赋给 property 属性指定的 Bean 属性。

5.6.3 <jsp:getProperty>

<jsp: getProperty>属性用来从指定的 Bean 中读取指定的属性值，并输出到页面中。该 Bean 必须具有 getXxx()方法。<jsp:getProperty>标识的使用格式如下。

```
<jsp:getProperty name = "BeanName" property = "propertyName"/>
```

name 属性：用来指定一个存在某 JSP 范围中的 Bean 实例。<jsp:getProperty>动作将会按照 page、request、session 和 application 的顺序来查找这个 Bean 实例，直到第一个实例被找到，若任何范围内不存在这个 Bean 实例则会抛出异常。

property 属性：指定了要获取由 name 属性指定的 Bean 中的哪个属性的值。例如它指定的值为"userName"，那么 Bean 中必须存在 getUserName()方法，否则会抛出异常。

5.6.4 JSP 动作元素示例

通过用户注册的示例，来具体了解这三个 JSP 动作的用法，创建 Web 项目 Ch05_1，在 src 创建包 com.bean，在该包中创建 User 类如示例 5-2，该 JavaBean 为封装数据的实体类，接下来在 WebRoot 下创建 register.jsp 文件。

示例5-5

```
/**
 * register.jsp 注册页面
 */
<%@ page language="java" contentType="text/html;charset=UTF-8" pageEncoding="UTF-
8"%>
<html>
<head>
<title>注册页面</title>
</head>
<body>
用户注册
<hr/>
<form action="registerSuccess.jsp" method="post" name="form1">
  用户名:<input type="text" name="userName"><br/>
  密   码:<input type="password" name="password" size="20"><br/>
  <input type="submit" value="注册" name="submit">
  <input type="reset" value="重置" name="reset">
</form>
</body>
</html>
```

registerSuccess.jsp

```
/**
 * registerSuccess.jsp 注册成功页面
 */
<%@ page language="java" contentType="text/html;charset=UTF-8" pageEncoding="UTF-
8"%>
<jsp:useBean id="user" scope="page" class="com.bean.User"/>
<jsp:setProperty name="user" property="*" />
<html>
<head>
<title>注册成功显示页面</title>
</head>
```

```
<body>
注册成功<hr>
使用 Bean 属性方法<br/>
用户名:<% =user.getName() %><br/>
密码:<% =user.getPassword() %>
<hr>
使用 getProperty:<br/>
用户名:<jsp:getProperty property="name" name="user"/><br/>
密码:<jsp:getProperty property="password" name="user"/>
</body>
</html>
```

该示例的显示效果如图 5-3、图 5-4 所示。



图 5-3 注册页面



图 5-4 注册成功页面

通过效果图,可以观察用户名没有获取到。我们来分析一下,在 register.jsp 页面中用户名的变量名为 userName,在 User.java 中用户名的属性为 name,所以在 registerSuccess.jsp 中,通过<jsp:useBean id="user" scope="page" class="com.bean.User"/>创建了一个 User 的实例,用<jsp:setProperty name="user" property="*" />给该实例的各属性赋值,可是提交的变量名与 user 属性名不一致,因此再去获取时,就获取不到值。因此 registerSuccess.jsp 应改为下面的形式。

```
/**
 * registerSuccess.jsp 注册成功页面
 */
<%@ page language="java" contentType="text/html;charset=UTF-8" pageEncoding="UTF-
8"%>
<jsp:useBean id="user" scope="page" class="com.bean.User"/>
<jsp:setProperty name="user" property="name" param="userName"/>
<jsp:setProperty name="user" property="password"/>
<html>
<head>
<title>注册成功显示页面</title>
</head>
<body>
注册成功<hr>
```

```
使用 Bean 属性方法<br/>
用户名：<% = user.getName() %><br/>
密码：<% = user.getPassword() %>
<hr>
使用 getProperty：<br/>
用户名：<jsp:getProperty property = "name" name = "user"/><br/>
密码：<jsp:getProperty property = "password" name = "user"/>
</body>
</html>
```

此时的效果如图 5-5 所示。



图 5-5 修改后注册成功页面

上例中若把<jsp:setProperty name="user" property="name" param="userName"/>改为<jsp:setProperty name="user" property="name" value="supperMan"/>，则效果如图 5-6 所示。由此就可以看出 value 与 param 的作用了。



图 5-6 修改后注册成功页面

5.7 上机练习

1. 编写封装数据的 JavaBean。

需求说明：编写通知公告发布系统封装数据的 JavaBean：User.java、Notice.java 和 Type.java。

实现思路：参照示例 5-2。

2. 编写封装业务的 JavaBean。

需求说明：编写通知公告发布系统封装逻辑的 JavaBean：UserControl. java、NoticeControl. java 和 TypeControl . java，注意连接数据库时利用工具类 ConnectionManager. java，每个业务逻辑可以只写查找全部内容的方法，在以后需要时再补充。

实现思路：参照示例 5-4。

3. JSP 页面调用 JavaBean。

需求说明：JSP 调用练习 2 封装业务逻辑的 JavaBean，在页面中显示所有的通知公告的类型(见图 5-7)。

实现思路：

提示代码：showType. jsp 中获取 Type 类型的 list 关键代码为

```
<%@ page language = "java"  import = "java.util.*,com.bean.*,com.dao.*,java.sql.*"
pageEncoding = "UTF-8"%>
......
<%
    TypeControl td  = new TypeControl ();
    List list = td.getAllTyge();
    if(list.size()>0){
        for(int i = 0;i<list.size();i++){
            Type type = (Type)list.get(i);
            out.print(type.getTypeName());
            out.println("<br/>");
        }
    }
%>......
```

://localhost:8080/Ch06_5/showType.jsp

通知公告类别
教学通知
科研通知
信息公告
招标公告

图 5-7 显示所有通知公告类型

4. 修改封装业务逻辑的 JavaBean，JSP 页面调用 JavaBean。

需求说明：showNoticeByType. jsp 调用的业务逻辑 NoticeControl. java 中的 showNoticeByType 方法，练习 2 中没有此方法，则修改 NoticeControl. java 增加此方法，再在 JSP 页面中调用，在页面中显示 Ntype 为 1 的所有信息。

实现思路：

提示代码：NoticeControl. java 中 getNoticeByType 方法关键代码为

```
public List getNoticeByType(int typeid){
      Connection con  =  null;
      PreparedStatement pStatement  =  null;
      ResultSet res  =  null;
```

```
        List list = new ArrayList();
        try {
            con = ConnectionManager.getConnection();
            String sql = "select * from Notice where Ntype=?";
            pStatement=con.prepareStatement(sql);
            pStatement.setInt(1,typeid);
            ResultSet rs=pStatement.executeQuery();
            while (rs.next()) {
                Notice notice = new Notice();
                    //把接收到的各属性值封装到一个 Notice 对象中
                notice.setContent(rs.getString("Ncontent"));
                notice.setCreateTime(rs.getDate("NcreateTime"));
                notice.setEditor(rs.getString("Neditor"));
                notice.setId(rs.getInt("Nno"));
                notice.setTitle(rs.getString("Ntitle"));
                notice.setType(rs.getInt("Ntype"));
                list.add(notice);
                }

        } catch (SQLException sqlE) {
            sqlE.printStackTrace();
        } finally {
            ConnectionManager.closeResultSet(res);
            ConnectionManager.closeStatement(pStatement);
            ConnectionManager.closeConnection(con);
        }
        return list;
    }
```

showNoticeByType.jsp 关键代码为

```
显示 Ntype=1 通知公告信息<br/>
<%
    NoticeControl noticeControl=new NoticeControl ();
    List list=noticeControl.getNoticeByType(1);
    if(list.size()>0){
        for(int i=0;i<list.size();i++){
            Notice notice=(Notice)list.get(i);
            out.print(notice. getTitle());
            out.println("<br/>");
        }
    }
%>
```

5. 修改封装业务逻辑的 JavaBean，JSP 页面调用 JavaBean。

需求说明：showNoticeById.jsp 调用的业务逻辑 NoticeControl.java 中的 showNoticeById 方法，练习 2 中没有此方法，则修改 NoticeControl.java 增加此方法，再在 JSP 页面中调用，在页面中显示 Nno 为 1 的所有信息。

实现思路：

提示代码：NoticeControl.java 中 showNoticeById 方法关键代码为

```
public Notice getNoticeById(int id){
        Connection con = null;
        PreparedStatement stm = null;
        ResultSet rs = null;
        Notice notice = new Notice();
        try {
            con = ConnectionManager.getConnection();
            String sql = "select * from Notice where Nno = " + id;
            stm = con.prepareStatement(sql);
            rs = stm.executeQuery();
            if(rs.next()){
                //把接收到的各属性值封装到一个 Notice 对象中
                notice.setContent(rs.getString("Ncontent"));
                notice.setCreateTime(rs.getDate("NcreateTime"));
                notice.setEditor(rs.getString("Neditor"));
                notice.setId(rs.getInt("Nno"));
                notice.setTitle(rs.getString("Ntitle"));
                notice.setType(rs.getInt("Ntype"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            ConnectionManager.closeResultSet(rs);
            ConnectionManager.closeStatement(stm);
            ConnectionManager.closeConnection(con);
        }
        return notice;
  }
```

showNoticeById.jsp 关键代码为

```
显示 Nno = 1 通知公告信息<br/>
<%
    NoticeControl noticeControl = new NoticeControl ();
    Notice notice = noticeControl.getNoticeById(1);
     %>
     标题：<% = notice.getTitle() %><br/>
     作者：<% = notice.getEditor () %><br/>
     时间：<% = notice.getCreateTime () %><br/>
     内容：<% = notice.getContent() %>
```

6. JSP 动作元素<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>。

需求说明：通过编写注册页面练习 JSP 动作元素<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>。

实现思路：参照 5.6 节。

5.8 总　　结

(1) JavaBean 在应用中主要负责封装数据和封装业务处理。

(2) JavaBean 的定义要遵循一定的规则，体现在以下几个方面：

- 公有类，并提供无参的公有的构造方法。
- 属性私有。
- 具有公有的访问属性的 getter 和 setter 方法。

(3) JSP 中动作元素与 JavaBean 密切相关的有<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>等。

5.9 作　　业

一、选择题

1. 以下对 JavaBean 的描述中正确的是(　　)。

A. JavaBean 最终是被保存在扩展名为 jsp 的文件中

B. 在 JSP 页面中只用通过<jsp:useBean>动作标识才可以调用 JavaBean

C. JavaBean 实质上就是一个 Java 类

D. 编译后的 JavaBean 放在项目的任何目录下，在 JSP 页面中都可以被调用

2. 设创建的 JavaBean 类中有一个 int 型的属性 number，下列哪个方法是设置该属性值的正确方法(　　)。

A.
```
public void setNumber(int n){
    number = n;
}
```

B.
```
void setNumber(int n){
    number = n;
}
```

C.
```
public void SetNumber(int n){
    number = n;
}
```

D.
```
public void Setnumber(int n){
    number = n;
}
```

二、简答题

1. 什么是 JavaBean?

2. 按功能 JavaBean 可分为哪几种?

3. 在 JSP 页面中如何使用 JavaBean?

三、程序题

1. 在 Web 项目中，创建一个 JavaBean，要求该 JavaBean 具有 name、age、sex 和 tel 的属性。

2. 在 Web 项目中，创建一个名为 ToUpper 的 JavaBean，用来转换传入的字符串参数为大写的。

第 6 章　JSP 内置对象

本章学习目标

- ✍ 掌握 JSP 的内置对象 request
- ✍ 掌握 JSP 的内置对象 response
- ✍ 掌握 JSP 的内置对象 out
- ✍ 掌握 JSP 的内置对象 session
- ✍ 掌握 JSP 的内置对象 application

到目前为止，我们已经了解了 JSP 的工作原理以及执行过程，还了解了 JSP 页面里包含的元素。这些是学习 JSP 的基础，为了方便 Web 程序的开发，在 JSP 页面中还设置了一些默认的对象，我们称之为内置对象。

6.1　什么是 JSP 内置对象

JSP 内置对象，就是当编写 JSP 页面时，无须做任何声明就可以直接使用的对象。如在示例 3-3 和示例 3-4 中出现了如下的代码片断。

```
<%
    int[ ] value = { 60, 70 ,80 };
    for (int i = 0;i < value.length; i++){
    out.println(value[i]);
%>
```

代码 out. println ()可以实现页面的输出显示，但是在代码中并没有任何方法声明或创建这个 out 对象，没有创建就可以直接使用的原因，就是因为 out 对象是 JSP 内置对象之一。

除了 out 对象以外，在 JSP 中还有其他一些内置对象。JSP 内置对象一共有 9 个：request、response、out、session、application、pageContext、config、page 和 exception。图 6-1 列举出了几个常用的内置对象，也是这章将要重点介绍的。

问题

为什么JSP的内置对象不需要实例化即可直接使用?

所谓内置对象就是由Web容器加载的一组类的实例,它不像一般的Java对象在创建类的实例时,必须要使用"new"关键字去构造,而是可以直接在JSP中使用的对象。特别要注意的是JSP的内置对象名称均是JSP的保留字,不得随便使用。

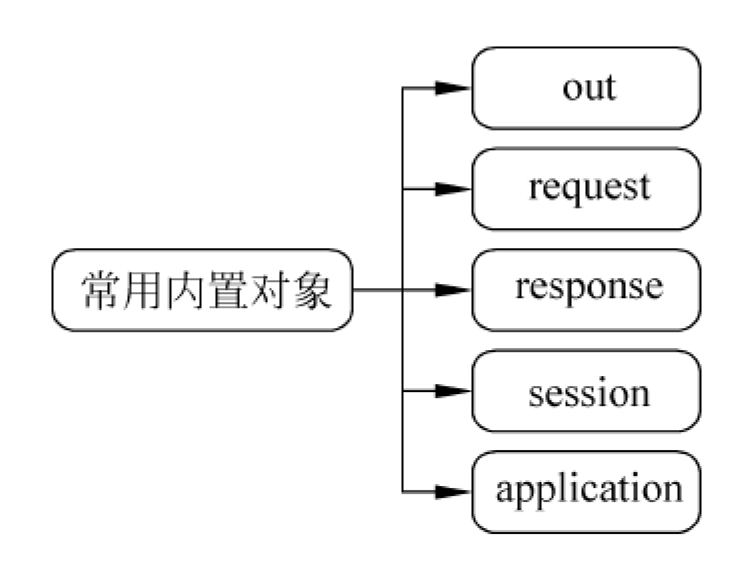


图 6-1 JSP常用的内置对象

6.2 JSP 内置对象 out

out 内置对象是在 JSP 开发过程中使用得最为频繁的对象,然而其使用起来也是最简单的。out 对象用于向客户端输出数据。out 对象常用的方法是 print ()和 println(),这两个方法都能用于在页面中打印出字符串信息,区别在于 println()输出信息后换行。比如要在页面上打印 hello JSP 我们可以这样写

```
<%
    out.print("hello JSP" );
%>
```

print ()方法只是将内容输出到屏幕上,其实,程序在处理时是先将内容放在缓冲区中,而不是直接输出,等到 JSP 引擎解释完程序后才把缓冲区中的数据输出到浏览器上。out 常用的方法还有: out.clear(),清除缓冲区中的数据,但不把数据写到客户端; out.newLine(),输出一个换行符。

6.3 JSP 内置对象 request

request 对象是 JSP 程序设计中最常使用的内置对象,它是 HttpServletRequest 类的实例,使用 request 对象可以对客户端在发出请求时传送给服务器的信息进行访问。request 对象主要用于处理客户端请求,其工作原理如图 6-2 所示。

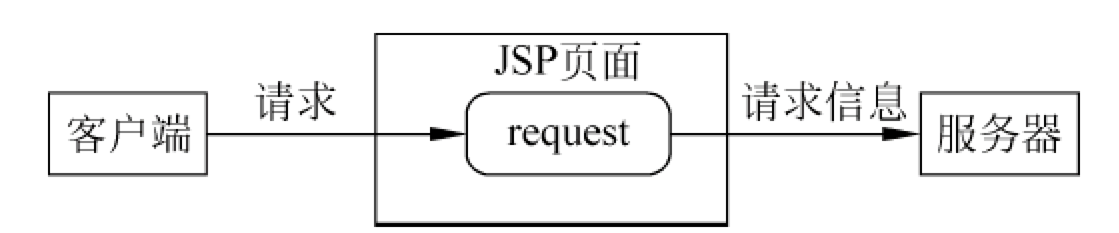


图 6-2 request 内置对象的工作原理

当客户端向服务器发起一个请求时,通常会把自身的一些信息,例如用户填写的表单数据,保存在客户端的 Cookie 信息等,一起发送给服务器。服务器将来自客户端的请求信息封装到 request 对象中,而被请求页面就可以使用 request 对象来获取这些信息。封装在请求中的信息通常简称为"请求信息",客户端发送给所请求页面的表单数据等通常被称为"请求参数"(parameter)。

request 对象的方法很多,下面让我们用表 6-1 列出该对象最常用的一些方法。

表 6-1 request 对象的几个常用的方法表

方 法 名 称	说 明
String getParameter(String name)	根据页面表单组件名称获取页面提交数据
String[ ] getParameterValues (String name)	获取一个页面表单组件对应多个值时的用户的请求数据
void setCharacterEncoding (String charset)	指定每个请求的编码,在调用 request. getParameter()方法之前进行设定,可以用于解决中文乱码问题
request. getRequestDispatcher(String path)	返回一个 javax. servlet. RequestDispatcher 对象,该对象的 forward 方法用于转发请求

了解了 request 对象具有的这些方法,下面我们一起来完成一个开发任务,通过这个任务来说明如何使用 request 对象的这些常用方法。开发任务的要求如下。

编程实现网站注册功能。注册信息包括:用户名、密码、你从哪里知道本网站的,如图 6-3 所示,页面提交后,显示输入的数据,如图 6-4 所示。

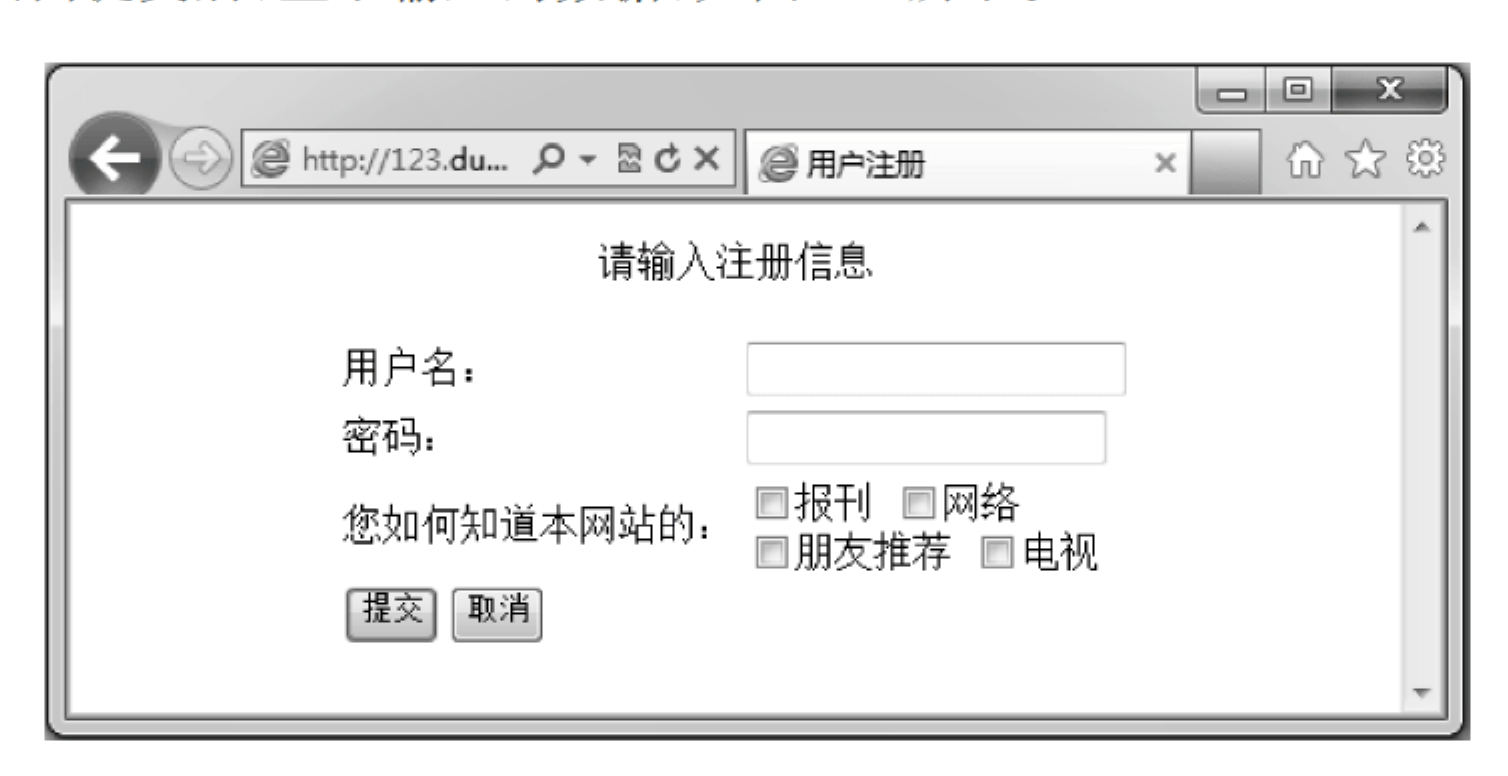


图 6-3 输入注册信息

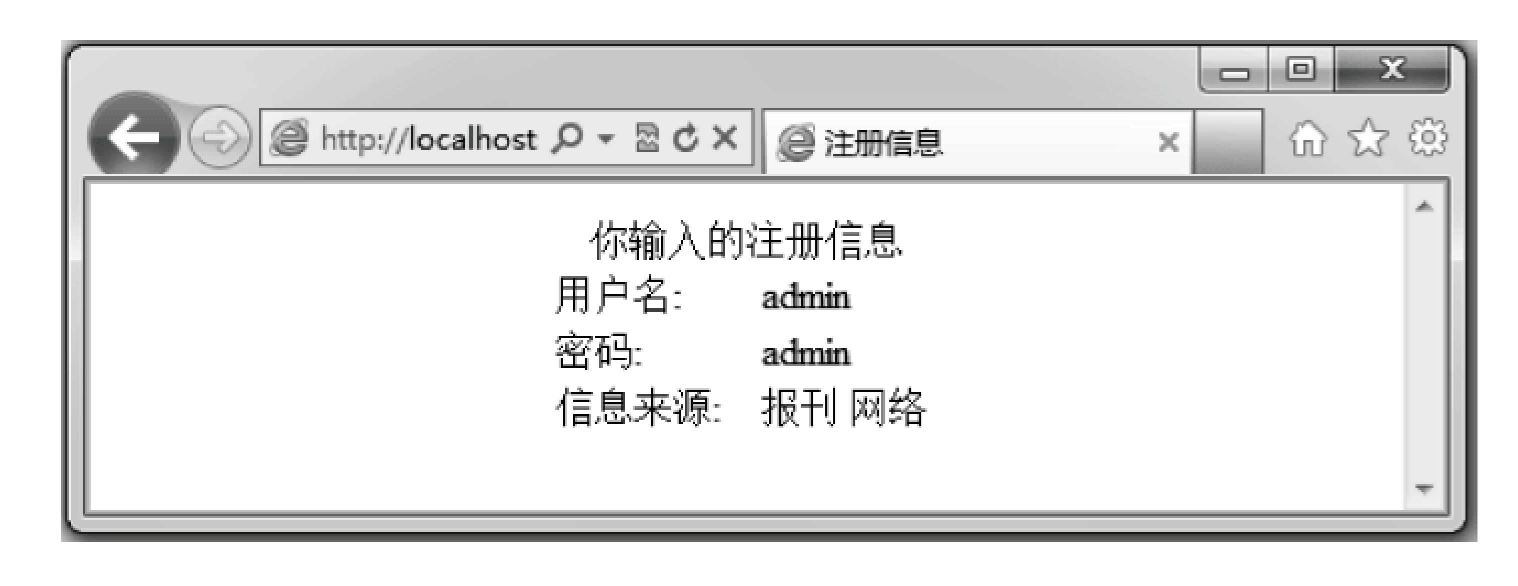


图 6-4 页面提交后现实的注册信息

实现代码如示例 6-1。

示例6－1

注册页面 reginput.jsp 如下。

```
<!--/**
 * 注册页面
 */
--><%@ page language="java" contentType="text/html; charset=utf-8" %>
<html>
    <head>
        <title>用户注册</title>
    </head>
    <body>
    <div align="center">请输入注册信息
        <form name="regForm" method="post" action="reginfo.jsp">
        <table border="0" align="center">
                <tr>
                <td>用户名：</td>
                <td><input type="text" name="name"></td>
                </tr>
                <tr>
                    <td height="19">密码：</td>
                    <td height="19"><input type="password" name="pwd"></td>
                </tr>
                <tr>
                <td>您如何知道本网站的：</td>
                <td>
                        <input type="checkbox" name="channel" value="报刊">报刊
                        <input type="checkbox" name="channel" value="网络">网络<br/>
                        <input type="checkbox" name="channel" value="朋友推荐">朋友
推荐
                        <input type="checkbox" name="channel" value="电视">电视
                    </td>
                </tr>
                <!-- 以下是提交、取消按钮 -->
                <tr>
                <td colspan="2" align="center">
                        <input type="submit" name="Submit" value="提交">
                        <input type="reset" name="Reset" value="取消">
                </td>
                </tr>
        </table>
        </form>
    </div>
    </body>
</html>
```

注册提交页面 reginfo.jsp 如下。

```
<!--/**
 * 注册提交页面
 */
--><%@ page language="java" contentType="text/html; charset=utf-8"%>
<%
    request.setCharacterEncoding("utf-8");
    String name = request.getParameter("name");
    String pwd = request.getParameter("pwd");
    String[ ] channels = request.getParameterValues("channel");
%>
<html>
    <head>
        <title>注册信息</title>
    </head>
    <body>
    <div align="center">你输入的注册信息
        <table border="0" align="center">
            <tr>
                <td width="80" height="20">用户名:</td>
                <td><%=name%></td>
            </tr>
            <tr>
                <td height="20">密码:</td>
                <td><%=pwd%></td>
            </tr>
            <tr>
                <td height="20">信息来源:</td>
                <td>
                <%
                    if (channels != null) {
                        for (int i = 0; i < channels.length; i++) {
                            out.print(channels[i] + " ");
                        }
                    }
                %>
                </td>
            </tr>
        </table>
    </div>
    </body>
</html>
```

代码说明：request 的 getParameter ()方法是最为常用的，使用此方法可以获得上一页面所提交的参数值。此外，注册页面(reginput.jsp)通过 HTML 表单给注册提交页面(reginfo.jsp)提交了两个参数，名称分别为 name 和 pwd，通过调用 request.getParameter("name")和 request.getParameter("pwd")就可以获取到这两个参数的值。另外，在注册页面中出现了一个复选框，所有复选项的名称都是 channel，这样我们在注册提交页面中使用 getParameterValues ("channel")方法就可以获取到一个数组，这个数组中存储的就是用

选中的复选项对应的值。



页面提交后，可能会出现中文乱码问题。这时，需要使用 request 对象的 setCharacterEncoding()方法，以指定每个请求的编码。在调用 request.getParameter( ) 之前调用该方法，并把字符集设定为 UTF-8，可以解决中文乱码问题。

6.4 JSP 内置对象 response

6.4.1 response 对象

response 对象也是 JSP 中最常使用的内置对象之一，它是 HttpServletResponse 类的实例。我们学习了如何使用 JSP 技术获取请求信息，那么 JSP 技术是怎样将服务器响应返回给客户端的呢？下面，我们就来讲解 JSP 内置对象 response 是如何实现用户响应的。其工作原理如图 6-5 所示。

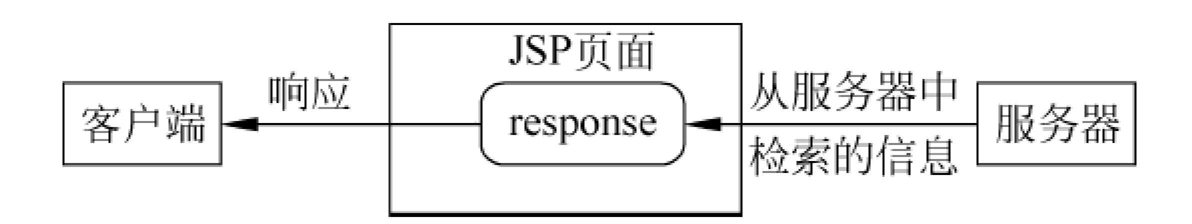


图 6-5 response 内置对象的工作原理

与 request 对象一样，response 对象也提供了多个方法用来处理 HTTP 响应，表 6-2 列出了几个常用的方法。

表 6-2 response 对象的几个常用的方法表

方法名称	说明
addCookie(Cookie cookie)	在客户端添加 Cookie
setContentType(String type)	设置 HTTP 响应的 contentType 类型
setCharacterEncoding(String charset)	设置响应所采用的字符编码类型
sendRedirect(String path)	将请求重新定位到一个不同的 URL 上

最常用的方法是 void sendRedirect (String path)。这个方法用来将请求重定向到一个不同的 URL 上。

下面我们通过完成一个开发任务，来说明 send Redirect ()方法的使用，这个开发任务的要求如下。在登录页面(login.jsp)上输入用户名、密码，提交至 control.jsp 进行处理，如果输入的用户名和密码都是 admin 跳转至欢迎页面(welcome.jsp)。登录页面如图 6-6 所示，成功跳转后的欢迎页面如图 6-7 所示。

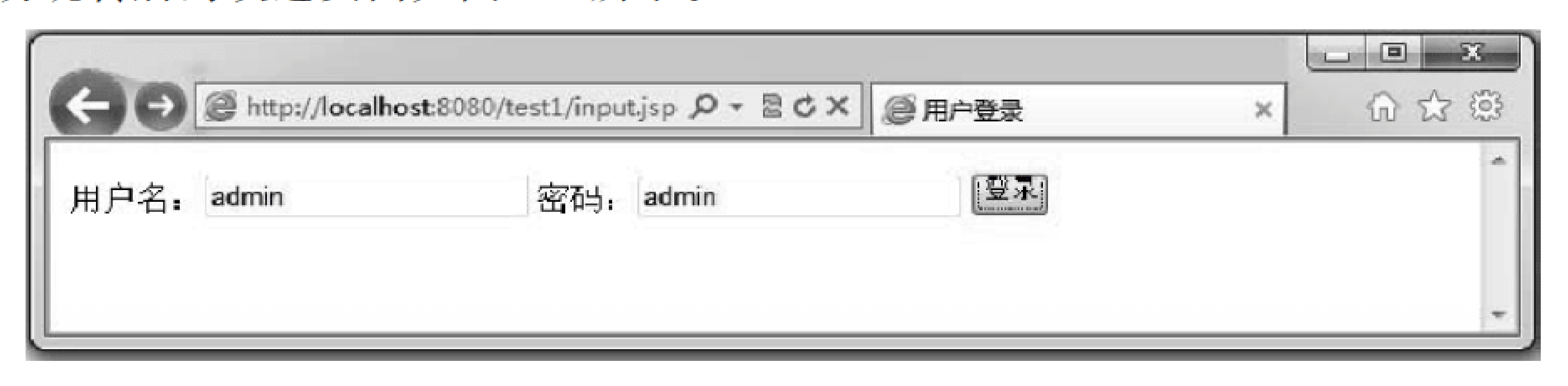


图 6-6 登录页面



图 6-7 欢迎页面

示例6-2

登录页面 login.jsp 如下。

```
<!--/**
 * 登录页面
 */
--><%@ page language="java" contentType="text/html; charset=UTF-8"%>
<html>
    <head>
        <title>用户登录</title>
    </head>
    <body>
        <form name="form1" method=post action="control.jsp">
            用户名:<input type="text" name="userName">
            密码:<input type="password" name="pwd">
            <input type="submit" value="登录">
        </form>
    </body>
```

登录处理页面 control.jsp 如下。

```
<!--/**
 * 登录处理页面
 */
--><%@ page language="java" contentType="text/html; charset=UTF-8"%>
<html>
    <head>
        <title>登录处理页面</title>
    </head>
    <body>
    <%
        request.setCharacterEncoding("UTF-8");
        String name = request.getParameter("userName");
        String pwd = request.getParameter("pwd");
        if(name.equals("admin")&& pwd.equals("admin")){
            response.sendRedirect("weclome.jsp");
        }
     %>
    </body>
```

欢迎页面 welcome.jsp 如下。

```
<!--/**
 * 欢迎页面
 */
--><%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
    <head>
        <title>欢迎</title>
    </head>
    <body>
        欢迎进入本页面!
    </body>
</html>
```

运行一下,你可能会注意到:由登录页面跳转至欢迎页面后,客户端重新建立了链接,URL 地址发生了改变,如图 6-7 所示。我们暂且不去探究为什么会这样,继续完善示例的功能。

问题

如果希望当登录成功后,在欢迎页面显示登录用户怎么办?

我们知道当用户提交请求后,使用 JSP 内置对象之一 request 对象可以获取到用户请求的数据,接下来我们要在 welcome.jsp 中显示用户名,所以修改 welcome.jsp 的代码,如示例 6-3 的代码所示。

示例6-3

```
<!--/**
 * 欢迎页面
 */
--><%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
    <head>
        <title>欢迎</title>
    </head>
    <body>
        <%
            String name = request.getParameter("userName");
         %>
        欢迎<%=name %>进入本页面!
      </body>
    </body>
</html>
```

查看运行效果,如图 6-8 所示。

从图 6-8 可以清晰地看到,原本应该显示用户名的位置,却显示为 null,与我们设想的结果完全不一样,这是什么原因呢? 大家不要着急,接下来我们就将解决这个问题。



图 6-8 读取用户信息

6.4.2 转发与重定向

首先对 control.jsp 的代码进行修改,修改后的代码如示例 6-4 所示。

示例6-4

```
<!--/**
 * 登录处理页面
 */
--><%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
    <head>
        <title>登录处理页面</title>
    </head>
    <body>
    <%
        request.setCharacterEncoding("UTF-8");
        String name = request.getParameter("userName");
        String pwd = request.getParameter("pwd");
         if(name.equals("admin")&& pwd.equals("admin")){ request.getRequestDispatcher
("weclome.jsp").forward(request,response);}
     %>
    </body>
</html>
```

运行示例 6-4 的代码,再次显示运行结果,如图 6-9 所示。

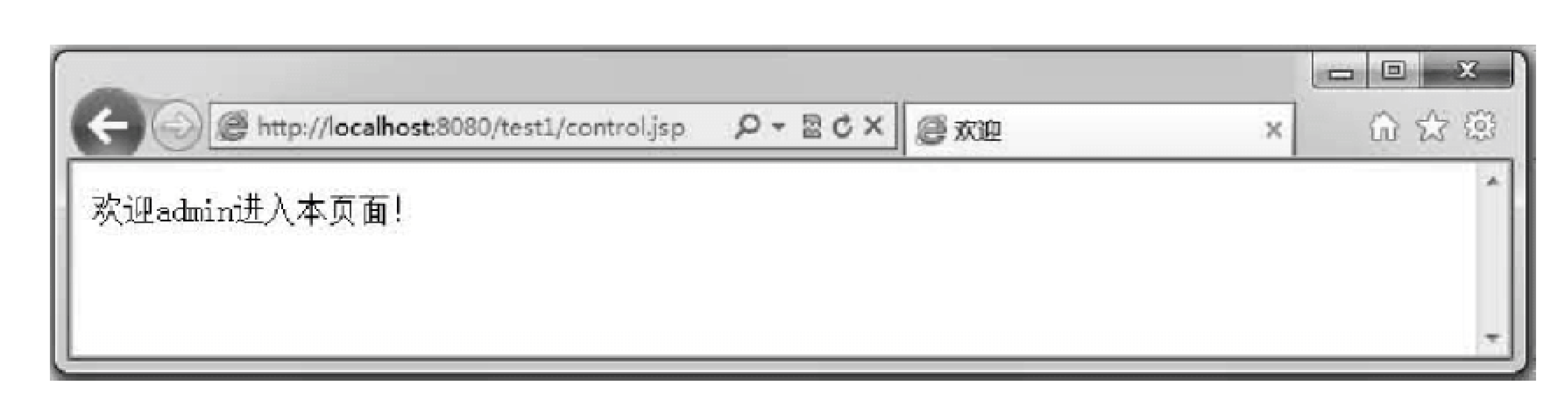


图 6-9 欢迎页面

经过修改代码,我们实现了再次获取用户信息,要想明白其中的奥妙,就必须要了解下面我们要讲的内容——JSP 页面的转发与重定向。

1. 转发

转发简单地说就是通过一个中介,将甲方的请求传递给乙方。从程序运行的角度来说

就是当客户端发送一个请求到服务器后，Web 服务器调用内部的方法在容器内部完成请求处理和转发动作，然后将目标资源发送给浏览器，整个过程都是在一个 Web 容器内完成，因而可以共享 request 范围内的数据。而对应到客户端，不管服务器内部如何处理，作为浏览器都只是提交了一个请求，因而客户端的 URL 地址不会发生改变。转发的实现很简单，使用 request 的 getRequestDispatcher ()方法即可实现。转发的作用：在多个页面交互过程中实现请求数据的共享。

2. 重定向

在示例 6-2 中，当用户登录成功后，我们使用的是 response 对象的 sendRedirect ()方法。那么该方法执行的结果是客户端重新向服务器请求一个地址链接，由于是发送新的请求，因而上次请求中的数据将随之丢失，我们将这种行为称为重定向。由于服务器重新定向了 URL，因而在客户端浏览器中显示的是新的 URL 地址，所以重定向可以理解为是浏览器至少提交了两次请求。



问题：转发和重定向有什么区别？

答：转发和重定向都能够实现页面的跳转，不同之处表现在以下几方面：

- 重定向过程：Web 服务器向浏览器发送一个新的 HTTP 请求，浏览器接受此响应后再发送一个新的 HTTP 请求到服务器，服务器根据此请求寻找资源并发送给叫览器。它可以重定向到任意 URL，不能共享 request 范围内的数据。重定向是在客户端发挥作用，通过请求新的地址实现页面转向。重定向是通过浏览器重新请求地址，在地址栏中可以显示转向后的地址。
- 转发过程：Web 服务器调用内部的方法在容器内部完成请求处理和转发动作，将目标资源发送给浏览器，它只能在同一个 Web 应用中使用，可以共享 request 范围内的数据。转发是在服务器端发挥作用，通过 forward 方法将提交信息在多个页面间进行传递。转发是在服务器内部控制权的转移，客户端浏览器的地址栏不会显示出转向后的地址。

6.5　JSP 内置对象 session

在讲 session 之前，我们先对 cookie 做一番介绍。

6.5.1　cookie 简介

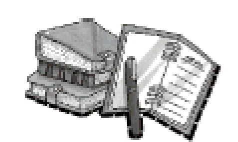
1. 为什么使用 cookie

我们都知道，浏览器与 Web 服务器之间是使用 HTTP 协议进行通信的，当某个用户发出页面请求后 Web 服务器进行响应，然后就断开了与浏览器的联系。因此当一个请求发送到 Web 服务器时，无论其是否是第一次访问，服务器都会把它当作第一次来对待。在实际开发中，我们往往希望服务器能够识别已访问过的用户。为了弥补 HTTP 协议的这个缺陷。Netscape 开发出了 cookie 这个有效的工具来保存某个用户的识别信息。

2. 什么是 cookie

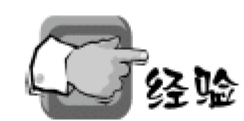
cookie 是由服务器端生成，发送给客户端浏览器的，浏览器会将其保存成某个目录下的

文本文件。当用户在浏览网站的时候 Web 服务器会将一些资料存放在客户端，这些资料包括用户在浏览网站期间输入的文字或是一些选择记录。当用户下一次访问该网站的时候，服务器会先从客户端查看是否有保留下来的 cookie 信息，然后依据 cookie 旧的内容，呈现特定的页面内容给用户。cookie 最典型的应用是判定注册用户是否已经登录网站，用户可能会得到提示，是否保存状态以便于在下一次进入系统时可以简化。另外，cookie 还会应用到"购物车"之类的业务处理中，用户可能会在购物网站中选择不同的商品，这些信息都可以使用 cookie 进行保存，然后在用户最终结算时进行信息提取。


资料

cookie 的作用表现在如下几个方面：

- 对特定对象的追踪，如访问者的访问次数、最后访问时间、路径等。
- 统计网页浏览次数。
- 在 cookie 有效期内，记录用户登录信息。
- 实现各种个性化服务，如针对不同用户喜好以不同的风格展示不同的内容。


经验

由于 cookie 会将用户的个人信息保存在客户端，例如用户名、计算机名、以往浏览和访问的网站等。这些信息中会包含一些很敏感的内容，尤其是用户私人的信息。所以从安全角度上，使用 cookie 存在着一定的风险。因此不建议在 cookie 中保存比较重要或敏感的内容。

3. 在 JSP 中使用 cookie

第一步：使用 page 指令导入类 javax. serverlet. http. cookie

```
<%@page import="javax.servlet.http.cookie"%>
```

第二步：创建 cookie 对象

我们通过调用构造函数 Cookie (String key,String value)创建新的 cookie 对象，其中，key 用于代表 cookie 的名称，value 用于表示当前 key 名称所对应的值。

第三步：写入 cookie

我们在学习 JSP 内置对象 response 对象常用方法时，有一个 addCookie ()方法.

```
response.addCookie(newCookie);
```

第四步：读取 cookie

读取时将会调用 request 对象的 getCookies ()方法，该方法将会返回一个 HTTP 请求头中的内容对应的 cookie 对象数组，因此必须要通过遍历的方式进行访问。

我们知道 cookie 是通过 key/value 方式进行保存的，因而在遍历数组时，需要通过调用 getName()对每个数组成员的名称进行检查，直至找到我们需要的 cookie，然后再调用 cookie 对象的 getValue()方法取得与名称对应的值。

示例6-5

登录页面 login. jsp 如下。

```
<!--/**
 * 登录页面
 */
--><%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登录页面</title>
  </head>
  <body>
    <form name="loginForm" method="post" action="doLogin.jsp">
            用户名:<input type="text" name="userName" />
            密码:<input type="password" name="pwd" />
            <input type="submit" value="登录">
        </form>
  </body>
</html>
```

登录处理页面 doLogin.jsp 如下。

```
<!--/**
 * 登录处理页面
 */
--><%@ page language="java" import="java.util.*,javax.servlet.http.cookie"
pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>处理登录页面</title>
  </head>
  <body>
    <%
        request.setCharacterEncoding("UTF-8");
        String name = request.getParameter("userName");
        String pwd = request.getParameter("pwd");
        if("admin".equals(name.trim())&& "admin".equals(pwd.trim())){
            //以 key/value 的形式创建 Cookie
            Cookie uname = new Cookie("uname", name.trim());
            response.addCookie(uname);
            response.sendRedirect("welcome.jsp");
        }
     %>
  </body>
</html>
```

欢迎页面 welcome.jsp 如下。

```
<!--/**
 * 欢迎页面
 */
--><%@ page language="java" import="java.util.*,javax.servlet.http.cookie"
pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>欢迎页面</title>
  </head>
  <body>
    <%
        //获取请求中的Cookie,以数组方式保存
        Cookie cookies[ ] = request.getCookies();
        //循环遍历数组,得到key为"uname"的Cookie
        for(int i=0;i<cookies.length;i++){
            Cookie ucookie = cookies[i];
            if(ucookie.getName().equals("uname"))//判断Cookie的名称
                //获取key对应的value,输出显示
                out.println("欢迎你: " + ucookie.getValue());
        }
     %>
  </body>
</html>
```

运行示例6-5的代码,显示运行结果,如图6-10及图6-11所示。



图6-10 示例6-5填写用户名、密码



图6-11 示例6-5显示结果

6.5.2 会话

就Web开发而言,一个会话就是用户通过浏览器与服务器之间进行的一次通话,它包含浏览器与服务器之间的多次请求、响应过程。简单地说就是在一段时间内,单个客户与Web服务器的一连串相关的交互过程。在一个会话中,客户可能会多次请求访问一个网页,也有可能请求访问各种不同的服务器资源。

如图 6-12 所示描述了浏览器与服务器的一次会话过程。当用户向服务器发出第一次请求时,服务器会为该用户创建唯一的会话,会话将一直延续到用户访问结束(浏览器关闭可以导致会话结束)。

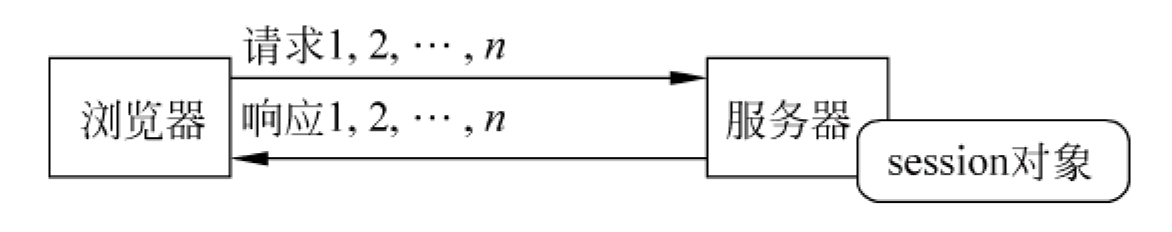


图 6-12 一次会话过程

JSP 提供了一个可以在多个请求之间持续有效的会话对象 session,session 对象允许用户存储和提取会话状态的信息。接下来,我们就来介绍 JSP 内置对象 session。

6.5.3 session 对象

session 对象是 javax.servlet.http.HttpSession 接口的实例对象,它封装了属于客户会话的所有信息。

session 在使用时遵守的是 session 机制。session 机制是一种服务器端的机制,在服务器端使用类似于散列表的结构来保存信息。当程序接收到客户端的请求时,服务器首先会检查这个客户端是否已经创建了 session。判断 session 是否创建是通过一个唯一的标识 sessionid 来实现的,如果在客户端请求中包含了一个 sessionid,则说明在此前已经为客户端创建了 session,服务器就会根据这个 sessionid 将对应的 session 读取出来。否则,就会重新创建一个新的 session,并生成一个与此 session 对应的 sessionid,然后将 sessionid 在本次响应的过程返回到客户端保存。

例如,我们在上网时,找到了一个下载网址,可是当单击"下载"条时,系统会自动转入登录页面,提示用户登录网站,当然如果你已经登录,就不会面临这样的问题了,根据前面提到的,系统即通过 session 实现对网站的访问控制。

问答

问题:sessionid 会返回到客户端,那么在客户端 sessionid 会保存在什么位置呢?

答:在客户端保存用户信息的是 cookie,因此保存 sessionid 的方式也是使用 cookie 来实现的。在客户端的 cookie 中,保存 sessionid 的名称类似于 SESSIONID,而 sessionid 的值也是一串复杂字符串,例如 jsessionid = 2A11D30C7B32329D7C8BF16DC598C509,其中 jsessionid 就是 sessionid 的名称,后面的字符串就是分配的 sessionid 对应的值。

下面,我们就来创建一个 sessionID,然后从 cookie 中读取 sessionID,对比两个 sessionID 是否相同。

示例6-6

在 create.jsp 页面中创建一个 session。

```
<!--/**
 * 在页面中创建 session
 */
```

```
--><%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@page import="javax.servlet.http.Cookie"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>addCookie.jsp</title>
  </head>
  <body>
    <%
        session.setAttribute("test","hello JSP");
        response.sendRedirect("getCookie.jsp");
     %>
  </body>
</html>
```

在 getCookie.jsp 中进行读取。

```
<!--/**
 * 在页面中读取 sessionID
 */
--><%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@page import="javax.servlet.http.Cookie"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>getCookie.jsp</title>
  </head>
  <body>
    <%
        out.print("sessionid:" + session.getId());
        out.print("<br/>");
        Cookie[] cookies = request.getCookies();
        if(cookies!= null){
            for(int i = 0;i<cookies.length;i++){
                out.print("cookie name:" + cookies[i].getName());
                out.print("<br/>");
                out.print("cookie value:" + cookies[i].getValue());
            }
        }
      %>
  </body>
</html>
```

示例运行的效果如图 6-13 所示。

由图 6-13 的输出结果可以看出，cookie 的值与 sessionid 的值是一致的，这也说明了，sessionid 被保存在 cookie 中，使用 cookie 的 getValue()方法就可以获取到 sessionid。

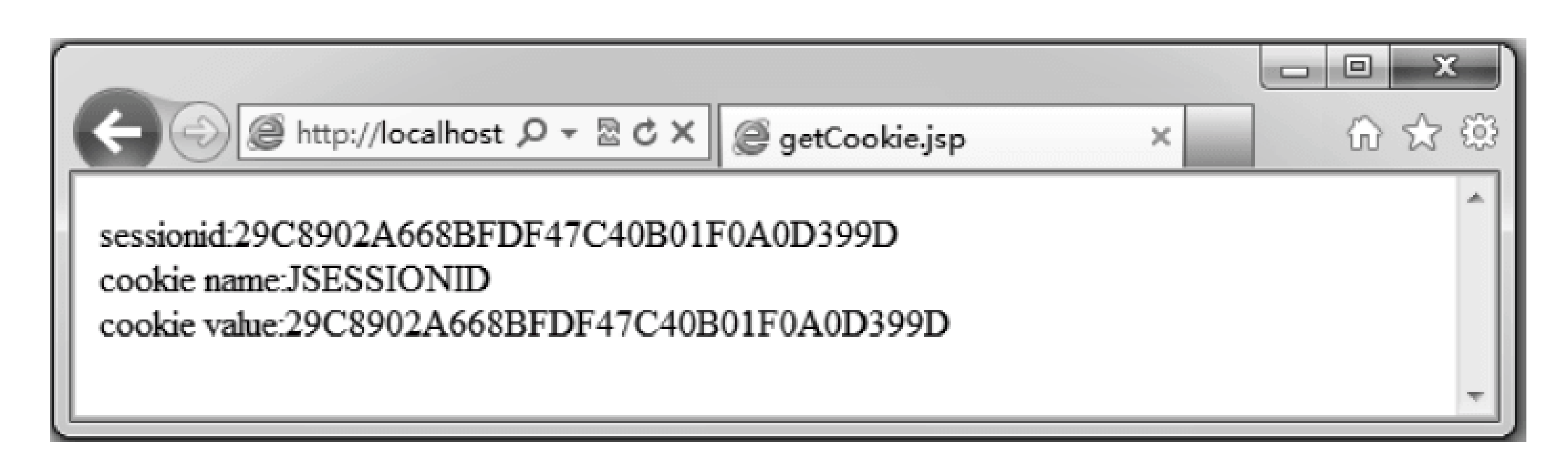


图 6-13 读取 sessionID



问题：如果不使用 response 进行重定向，而是使用 getRequestDispatcher( ).forward(request,response)方法也可以吗？

答：不可以，因为使用 response 对象的 sendRedirect()方法是将页面重定向到一个新的地址，即重新向服务器发送了一个请求，服务器已经对上一个请求做出了处理，在客户端写入了 cookie。如果使用转发的形式，那么服务器接收的是相同的请求，并没有返回响应，因而在客户端没有写入 cookie。

对比

session 与 cookie 均能实现信息的保存，但是二者的区别如下：

- session 是在服务器端保存用户信息，cookie 是在客户端保存用户信息。
- session 中保存的是对象，cookie 保存的是字符串。
- session 对象随会话结束而关闭，cookie 可以长期保存在客户端。
- cookie 通常用于保存不重要的用户信息，重要的信息使用 session 保存。

6.5.4 使用 session 实现权限控制

在 JSP 中 session 对象用来存储有关用户会话的所有信息。一个用户对应一个 session，并且随着用户的离开 session 中的信息也会消失。在 JSP 中谈到访问控制，其实现就是基于 session 对象来完成的。

session 对象的常用方法如表 6-3 所示。

表 6-3 session 对象的几个常用的方法

方法名称	说明
void setAttribute(String key,Object value)	以 key/value 的形式保存对象值
Object getAtrribute(String key)	通过 key 获取对象值
void invalidate()	设置 session 对象失效
String getId()	获取 session id
void setMaxInactiveInterval(int interval)	设定 session 的非活动时间
Int getMaxInactiveInterval()	获取 session 的有效非活动时间，以秒为单位

接下来，就使用 session 为通知公告发布系统-后台增加访问控制。当没有增加访问控制功能时，只要知道后台网页地址，在 IE 地址栏中直接写入地址即可发布信息，这是非常可

怕的。当我们在程序中利用 session 增加访问控制，再在地址栏中输出发布网页地址，则直接跳到登录页面。完成此功能只需在后台页面中增加如下小脚本代码。

示例6-7

```
<!--/**
 * 在后台页面中增加小脚本,控制访问
 */
-->
<%@ page language="java" import="java.util.*,com.bean.*" pageEncoding="UTF-8"%>
……
    <%
    User user = (User) session.getAttribute("LOGINED_USER");
    if (user == null) {
        response.sendRedirect("../../login.jsp");
    }else{
     %>
        ……

    <%
    }
%>
```

通过 session.getAttribute()方法，查找在 session 中是否有 key 为 LOGINED_USER 的用户，如果没有则重定向到 login.jsp 页面。那么在处理登录的代码中应该有

```
request.getSession().setAttribute("LOGINED_USER",user)
```

把登录成功的 user 放到 key 为 LOGINED_USER 的 session 中。

6.6 JSP 内置对象 application

我们刚刚学习完 session 对象，session 可以保存当前每个用户的会话状态信息，一个用户对应着一个 session。但是如果存在这样一个数据（如应用的访问人数），需要 Web 应用系统中的所有用户共享，该如何实现呢？可以使用 JSP 的另一个内置对象 application。

application 对象类似于系统的“全局变量”用于实现用户之间的数据共享，且只有一个实例。application 对象的常用方法如表 6-4 所示。

表 6-4 application 对象的几个常用的方法

方 法 名 称	说 明
void setAttribute(String key,Object value)	以 key/value 的形式保存对象值
Object getAtrribute(String key)	通过 key 获取对象值
String getAttribute(String path)	返回相对路径的真实路径

那么 application 对象该如何使用呢？接下来我们通过一个开发任务来具体说明 application 的用法。

在网站系统中统计并显示已访问的人数,该如何实现呢?

示例6-8

loginWeb.jsp 页面。

```
<!--/**
 * 在页面中登录后 application 增加 1
 */
--><%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>login.jsp</title>
  </head>
  <%
      Integer count = (Integer)application.getAttribute("count");
      if(count != null){
          count = 1 + count;
      }else{
          count = 1;
      }
      application.setAttribute("count",count);
 %>

  <body>
    <form name="loginForm" method="post" action="showCount.jsp">
            用户名:<input type="text" name="userName" />
            密码:<input type="password" name="pwd" />
            <input type="submit" value="登录">
        </form>
  </body>
</html>
```

showCount.jsp 页面。

```
<!--/**
 * 在页面中显示统计人数
 */
--><%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
      Integer i = (Integer)application.getAttribute("count");
      out.println("统计访问量:目前有 " + i +" 个人访问过本网站" );
 %>
```

示例运行的效果如图 6-14 所示。

至此,JSP 的几个常用的内置对象都已经介绍给大家了,下面通过表 6-5 对这些内置对象进行一个简要总结。

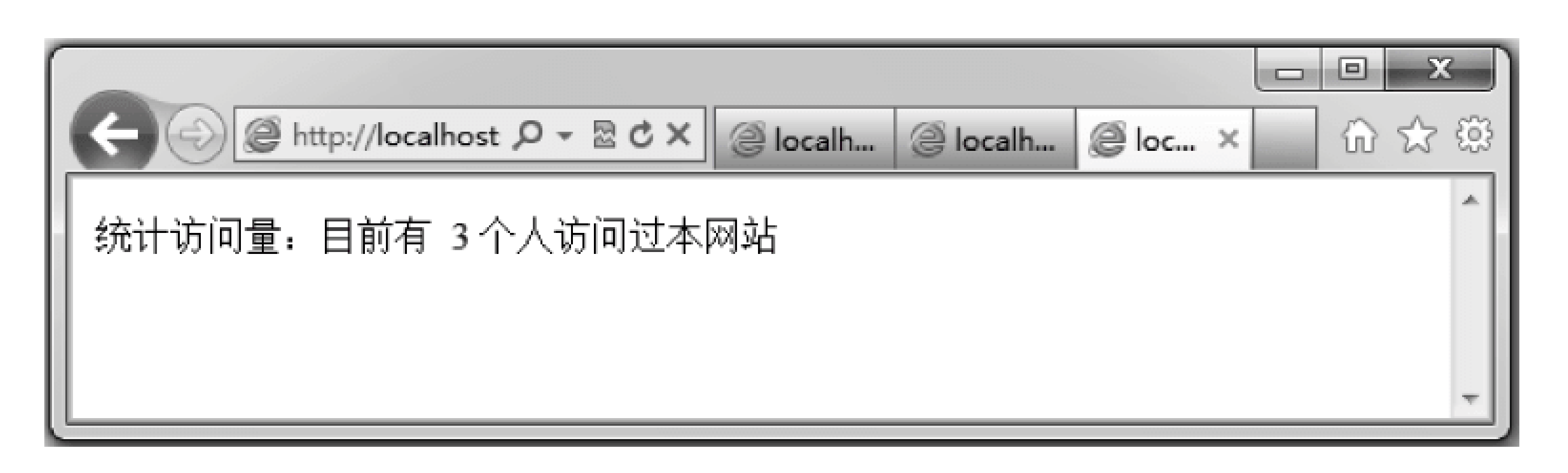
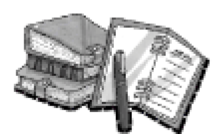


图 6-14 已访问人数统计页面

表 6-5 常用内置对象总结

内置对象名称	说　明
out 对象	用于向客户端输出数据
request 对象	主要用于客户端的请求处理
response 对象	用于响应客户端的请求并向客户端输出信息
session 对象	用来储存有关用户会话的所有信息
application 对象	类似于全局变量用于实现用户之间的数据共享


资料

JSP 的其他内置对象如下：

- pageContext：提供访问其他隐含对象的方法。pageContext 对象的常用方法如下：
  getRequest( )：获得 request 对象；
  getResponse( ) ：获得 response 对象；
  getSession( )：获得 session 对象；
  getOut()：获得 out 对象；
  setAttribute( )：保存属性；
  getAttrubute( )：获得属性；
  indude( )：包含其主页面在 pageContext 对象中保存的属性，只能在当前页面中去获取。
- page：表示当前页面。类似于 Java 中的 this。在 JSP 页面中，很少使用 page 对象。
- config：用于存放 JSP 编译后的初始数据。与 page 对象一样，在 JSP 页面中很少使用。
- exception：表示 JSP 页面运行时产生的异常，该对象只有在错误页面(page 指令中设定 isErrorPage 为 true 的页面)中才能够使用。

6.7 对象范围

在 JSP 页面中的对象，无论是用户创建的对象，还是 JSP 的内置对象，都有一个范围，范围定义了什么情况下，哪些 JSP 页面可以访问这些对象。这个概念类似于 Java 中的实例变量、局部变量和类变量的概念。在 JSP 中，对象有四种范围 page、request、session 和 application。

6.7.1 page 范围

page 范围指单一 JSP 页面的范围，page 范围内的对象只能在创建对象的页面中访问。在 page 范围内可以使用 pageContext 对象的 setAttribute ()和 getAttribute ()方法来访问具有这种范围类型的对象。page 范围内的对象在客户端每次请求 JSP 页面时创建，在服务器发送响应或请求转发到其他的页面或资源后失效，见示例 6-9。

示例6-9

```
<!--/**
 * page 范围 test1.jsp
 */
--><%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
     <title>PageScope</title>
  </head>
  <body>
    <%
    String name = "admin";
        pageContext.setAttribute("name",name);
     %>
     test1:<% = pageContext.getAttribute("name") %>
     <br/>
     <%
     pageContext.include("test2.jsp");
      %>
  </body>
</html>
```

test2.jsp 页面代码为

```
test2:<% = pageContext.getAttribute("name") %>
```

运行效果如图 6-15 所示。

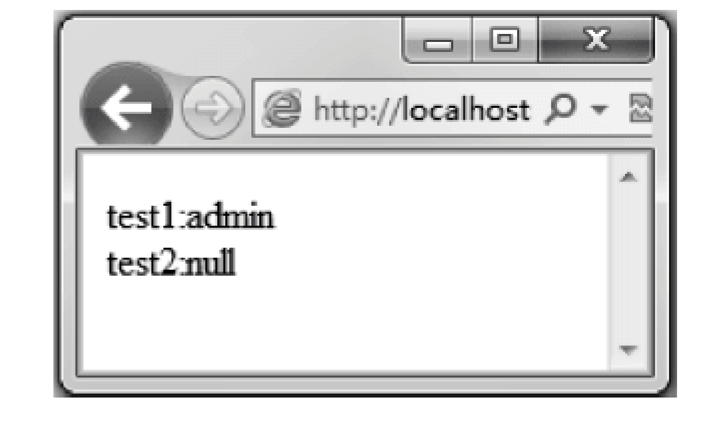


图 6-15 page 范围效果图



资料

pageContext 对象本身也属于 page 范围，具有 page 范围的对象被绑定到 pageContext 对象中。

6.7.2 request 范围

相对于在 page 范围内的对象与 pageContext 绑定在一起，request 范围内的对象则是与户端用户的请求绑定在一起，即 request 范围内的对象在页面转发或包含中有效。在范围内的对象同样可以通过调用 request 对象的 setAttribute ()与 getAttribute ()方法找到。同时在调用 forward ()方法转向的页面或者调用 include ()方法包含的页面时，都可以访问

request 范围内的对象。需要注意的是，因为请求对象对于每次客户端的用户请求是不同的，所以对于任何一个新的请求，都要重新创建该范围内的对象。而当请求结束后，创建的对象也随之失效，见示例 6-10。

示例6－10

test3.jsp 为

```
<!--/**
 * request 范围 test3.jsp
 */
--><%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
     <title>RequestScope</title>
  </head>
  <body>
    <%
    String name = "admin";
    request.setAttribute("name",name);
     %>
     test3:<% = request.getAttribute("name") %>
     <br/>
     <%
        pageContext.include("test4.jsp");
      %>
  </body>
</html>
```

test4.jsp 页面代码为

```
test4:<% =request.getAttribute("name") %>
```

运行效果如图 6-16 所示。

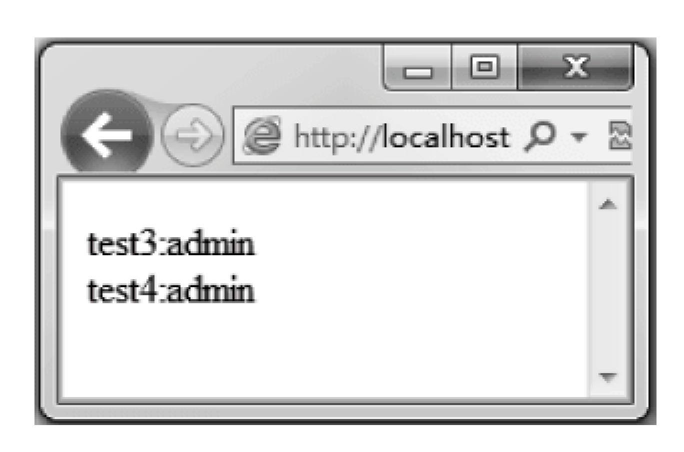


图 6-16 request 范围效果图

可以调用 request 对象的 setAttribute()方法，将 String 对象保存到 request 范围，然后调用 request 对象的 getAttribute ()方法访问 request 范围中的对象。

6.7.3 session 范围

JSP 容器为每一次会话创建一个 session 对象。在会话期间，只要将对象绑定到

session 中，对象的范围就为 session。在会话有效期间都可以访问 session 范围内的 session 对象。代码如示例 6-11 所示。

示例6－11

test5. jsp 为

```
<! --/* *
 * session 范围 test5. jsp
 */
 -->
 ……
        <%
        String req = "request";
        String ses = "session";
        request.setAttribute("req",req);
        session.setAttribute("ses",ses);
        response.sendRedirect("test6.jsp");
         %>
 ……
```

test6. jsp 页面代码为

```
request:<% = request.getAttribute("req") %><br/>
session:<% = session.getAttribute("ses") %>
```

运行效果如图 6-17 所示。

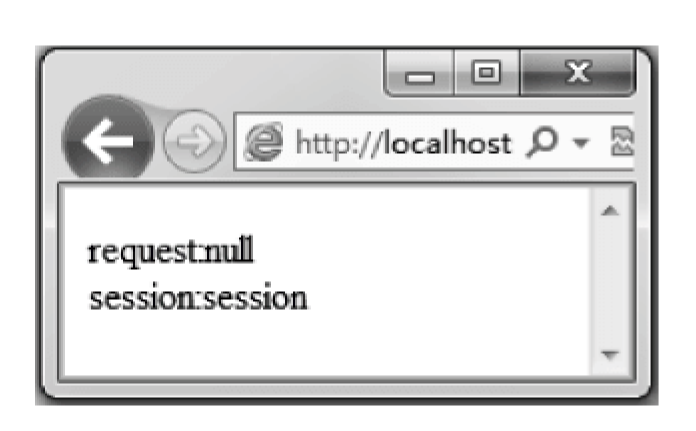


图 6-17 session 范围效果图

使用 request 对象将页面重定向到 test6. jsp，在 test6. jsp 中能够读取到 session 对象，由此可见 session 范围内的对象在会话有效期内有效，使用 response. sendRedirect ()重定向到另外一个页面的时候，相当于重新发起了一次请求，而上一次请求中的 request 对象则随之失效象。

6.7.4 application 范围

相对于 session 范围针对一个会话，application 的范围则面对整个 Web 应用程序，服务器启动后就会创建一个 applicatlon 对象，被所有用户所共享，其范围最大。application 对象也具有 setattribute ()和 getAttribute()方法，用于对该范围内的对象进行存储访问。

示例6－12

test7. jsp 为

```
<! --/* *
 * application 范围 test7.jsp
 */
-->
……
    <%
    String ses = "session";
    String app = "application";
    session.setAttribute("ses",ses);
    application.setAttribute("app",app);
    response.sendRedirect("test8.jsp");
    %>
……
```

test6.jsp 页面代码为

```
session:<% = session.getAttribute("ses") %><br/>
application:<% = application.getAttribute("app") %>
```

运行效果如图 6-18 所示。

当关闭浏览器,再次运行 test6.jsp 时,结果如图 6-19 所示。

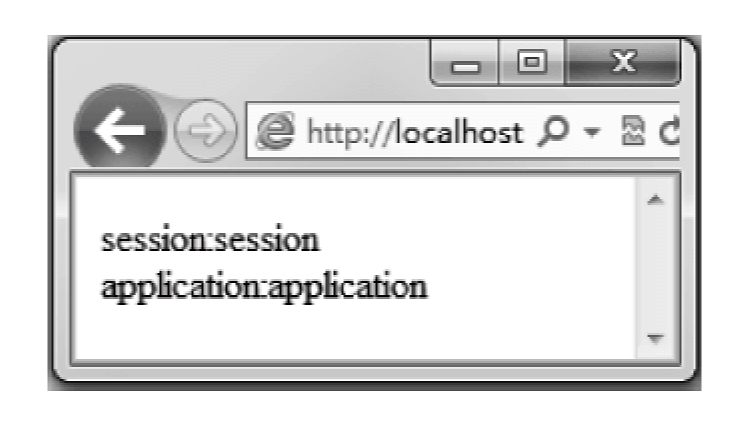


图 6-18 application 范围效果图

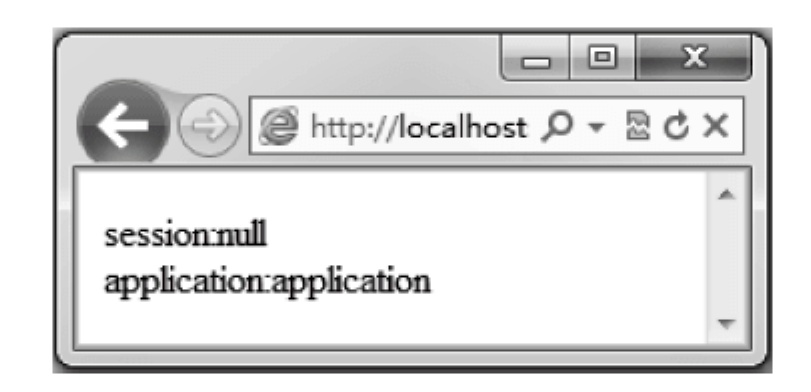


图 6-19 application 范围效果图

由此可见 session 范围针对的是一次会话,当浏览器关闭后会话也结束,所以无法读取。而 application 范围针对的是整个系统的服务,因而数据可以被再次读取,除非 Tomcat 重新启动。

6.8 上机练习

1. 转发、重定向、cookie。

需求说明:使用转发、重定向、cookie 实现登录。

实现思路:参照示例 6-4、示例 6-5。

2. 使用 session 实现访问控制。

需求说明:通知公告发布系统的后台页面只允许管理员登录后进入。

实现思路:参照示例 6-7。

3. 实现网站计数器功能

需求说明:在通知公告发布系统的前台每个通知公告显示页面的尾部,增加显示访问的人数统计,每当用户访问页面时,计数器加 1。

实现思路：参照示例 6-8。



图 6-20 网站访问量统计

4. 实现通知公告类型的动态读出。

需求说明：在通知公告发布系统的前台左部，通知公告类型名称。

实现思路：参照第 5 章的上机练习 3。



图 6-21 通知公告类型的动态输出

5. 实现根据类型动态读出通知公告列表。

需求说明：在通知公告发布系统的前台，通过连接通知公告类型名称，在页面右部动态读出属于该类型的通知列表，并显示。

实现思路：参照第 5 章上机练习 4。

关键代码提示：

（1）foreground.jsp 页面中通知通告类型链接为

```
<a href = "page/foreground/showNoticeByType.jsp?typeid = <% = type.getId() %>"
target = "showNotice">
```

```
<%
        typeControl typeControl = new typeControl ();
        List list = typeControl.getAllType();
        if(list.size()> 0){
                for(int i = 0;i< list.size();i++){
                Type type = (Type)list.get(i);
 %>
<a href = "showNoticeByType.jsp?typeid = <% = type.getId()  %>" target = "showNotice">
        <% = type.getTypeName() %></a>< br/>
<%
                }
        }
 %>
```

（2）showNoticeByType.jsp 的关键代码提示：

```
<%
        int id = Integer.parseInt(request.getParameter("typeid"));
        NoticeControl noticeControl = new NoticeControl();
        List list = noticeControl.getNoticeByTypeId(id);
        if(list.size()> 0){
                for(int i = 0;i< list.size();i++){
                Notice notice = (Notice)list.get(i);
 %>
        <a href = "#" target = "showNotice">
        <% = notice.getTitle() %></a>< br/>
        <%
                }
        }
 %>
```

效果如图 6-22 所示。

6. 实现动态显示通知公告的详细信息。

需求说明：在通知公告发布系统的前台，通过连接页面右部动态通知列表的信息，在右部显示该通知的详细信息。

实现思路：参照第 5 章上机练习 5。



图 6-22 根据类型动态读出通知公告列表



图 6-23 通知公告详细信息的动态输出

关键代码提示：

(1）首先在 showNoticeByType.jsp 页面中<a href="#" target="showNotice">改为

```
<a href="page/foreground/showNoticeDetailByID.jsp?noticeid=<%=notice.getId()%>"
target="showNotice">
```

（2）showNoticeDetailByID.jsp 的关键代码提示：

```
<!--/**
 * 显示 Notice 列表 JSP 页面代码提示
 */
-->
......
  <%
    request.setCharacterEncoding("utf-8");
    int noticeid = Integer.parseInt(request.getParameter("noticeid"));
    NoticeControl noticeControl = new NoticeControl();
    Notice notice = noticeControl.getNoticeById(noticeid);

%>
    <table>


        <tr>
            <td>标题：</td><td><%=notice.getTitle()%></td>
        </tr>
        <tr>
            <td>作者：</td><td><%=notice.getEditor()%></td>
        </tr>

        <tr>
            <td>时间：</td><td><%=notice.getCreateTime()%></td>
        </tr>
        <tr>
            <td>内容：</td><td><%=notice.getContent()%></td>
        </tr>
    </table>......
```

6.9 总　　结

（1）JSP 内置对象，就是当编写 JSP 页面时，无须做任何声明就可以直接使用的对象。

（2）内置对象 request 可以对客户端在发出请求时传送给服务器的信息进行访问。request 对象主要用于处理客户端请求。

（3）内置对象 response 用来处理 HTTP 响应。

（4）转发就是当客户端发送一个请求到服务器后，Web 服务器调用内部的方法在容器内部完成请求处理和转发动作，然后将目标资源发送给浏览器。使用 request 的 getRequestDispatcher()方法即可实现。重定向是客户端重新向服务器请求一个地址链接。使用 request.getRequestDispatcher(URL).forward(request,response)方法实现。

(5) 内置对象 session 对象封装了属于客户会话的所有信息。

(6) 内置对象 application 对象类似于系统的"全局变量"用于实现用户之间的数据共享,且只有一个实例。

(7) cookie 是由服务器端生成,发送给客户端浏览器的,浏览器会将其保存成某个目录下的文本文件。

(8) 通过 cookie 可以实现浏览器与服务器之间的数据传递。

(9) 就 Web 开发来说,一个会话就是用户通过浏览器与服务器之间的一次通话,包含浏览器与服务器之间的多次请求、响应过程。

(10) session 与 cookie 均能实现信息的保存,但是二者的区别如下:

- session 是在服务器端保存用户信息,cookie 是在客户端保存用户信息。
- session 中保存的是对象,cookie 保存的是字符串。
- session 对象随会话结束而关闭,cookie 可以长期保存在客户端。
- cookie 通常用于保存不重要的用户信息,重要的信息使用 session 保存。

6.10 作 业

一、选择题

1. 把一个用户名为"Tom"存在 session 对象中,则下列语句正确的是(   )。

A. session.setAttribute(name,Tom)　　B. session.setAttribute("name","Tom")

C. session.setAttribute(Tom,name)　　D. session.setAttribute("Tom","name")

2. 运行如下 JSP 代码,以下说法正确的是(   )。

```
<%
    String str = "hello JSP";
    session.setAttribute("title",str) ;
    String getStr = session.getAttribute("title");
    out.println(getStr);
%>
```

A. 运行成功,页面输出 hello JSP

B. 运行成功,页面输出 title

C. 代码 session.setAttribute("title",str) 有错,无法运行

D. 代码 String getStr = session.getAttribute("title") 有错,无法运行

3. JSP 提供了一个可以在多个请求之间持续有效的内置对象是(   ),该对象与浏览器一一对应。

A. request　　B. response　　C. session　　D. application

二、简答题

1. JSP 的内置对象有哪些?简要说明这些内置对象在 JSP 中的主要功能?

2. 重定向与转发的区别是什么?

3. 什么是 cookie?其作用是什么?

三、程序题

1. 在 one.jsp 中设置一个超链接,链接值 two.jsp 文件,并向 two.jsp 传递两个参数 a 和 b,在 two.jsp 中输出这两个值。

2. 编写一个 JSP 页面,统计该网页被访问的次数。

第 7 章　Servlet 技术

本章学习目标

- 掌握什么是 Servlet
- 了解 Servlet 与 JSP 的关系
- 掌握 Servlet 的生命周期
- 了解 Servlet API
- 掌握使用 Servlet 作为控制器

通过前面章节的学习，已经了解了 JSP 技术的体系结构和技术内容等知识。在 Internet 上，客户端通过使用 HTTP 协议，向服务器端发送请求信息，服务器对请求数据进行处理，并把处理后的结果响应给客户端。那么这一切是怎么做到的呢？Servlet 是什么技术，能解决哪些事情。本章我们将逐步学习 Servlet 的相关技术。

7.1　Servlet　简　介

使用 JSP 技术开发 Web 程序，是在 JSP 中写入 Java 代码，当服务器运行 JSP 时，执行 Java 代码，动态获取数据，并生成 HTML 代码，最终显示在客户端浏览器上。整个过程如图 7-1 所示。

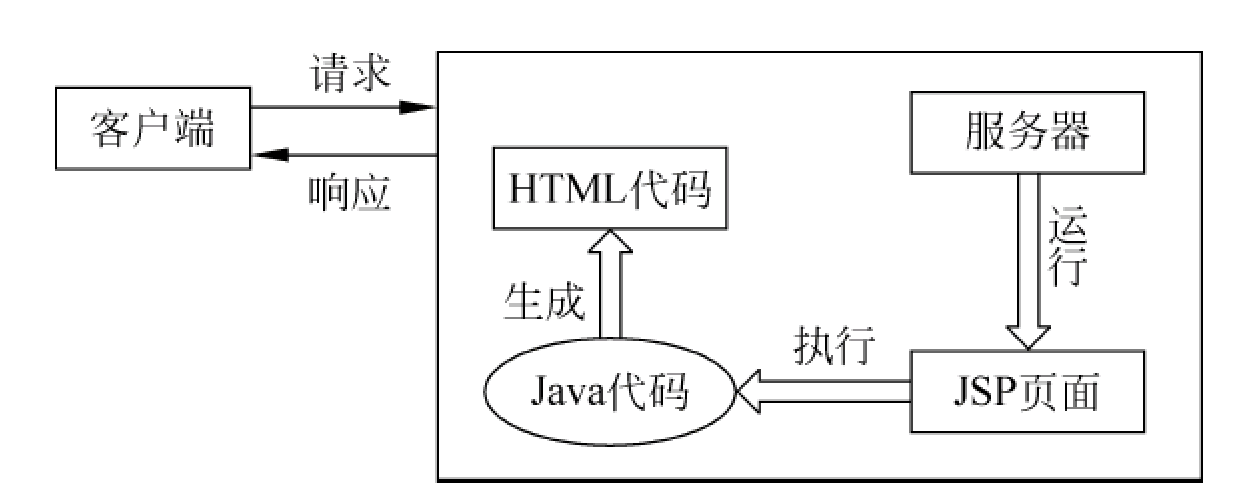


图 7-1　使用 JSP 技术开发 WEB 程序

问答

问题：在 JSP 技术出现之前，如何使用 Java 语言来编写 Web 程序？

答：在 JSP 技术出现之前，如果想动态生成 HTML 页面，那就只有在服务器端运行 Java 程序，并生成 HTML 格式的内容。运行在服务器端的 Java 程序就是 Servlet。过程如图 7-2 所示。

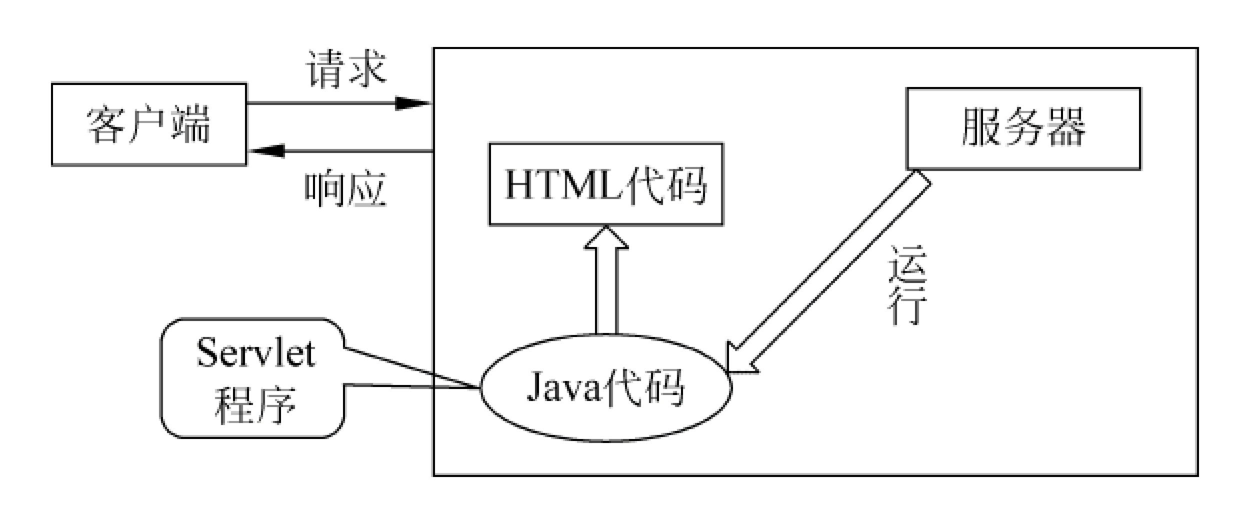


图 7-2 使用 Servlet 技术开发 Web 程序

那什么是 Servlet 呢? Servlet 是一个符合特定规范的 Java 程序,在服务器端运行,处理客户端请求响应,如图 7-3 所示。

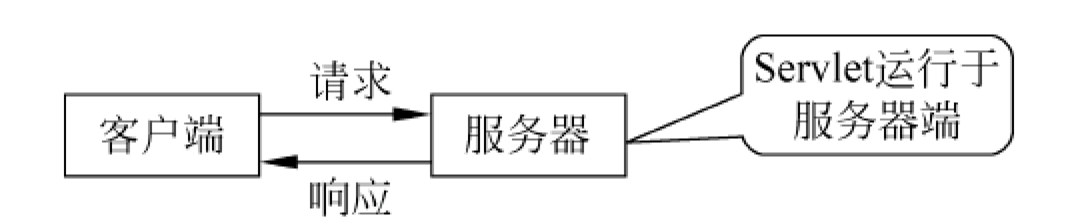


图 7-3 Servlet 运行于服务器端

尽管 Servlet 能够响应任何类型的请求,但在绝大多数的网络应用中,都是客户 HTTP 协议访问服务器端的资源,而我们所编写的 Servlet 也是应用于 HTTP 协议的响应,因此我们讲解的重点也将放在和这方面有关的 HttpServlet 类。

7.2 初 识 Servlet

了解了 Servlet 的功能和特点,知道了 Servlet 的定义,那么 Servlet 到底是什么样子? 符合哪些规范的 Java 程序才是 Servlet 呢?

下面就让我们来认识一下 Servlet。首先创建一个 Web 项目 Ch07_1,在 src 中创建包 com.servlet,在 com.servlet 包中创建名为 HelloServletTest.java 的 Servlet 文件,代码如示例 7-1 所示。

示例7—1

```
/**
 * 初识 Servlet
 */
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServletTest extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset = UTF - 8");
        PrintWriter out = response.getWriter();
```

```
            out.println("<html>");
            out.println(" <head><title>Servlet</title></head>");
            out.println(" <body>");
            out.println("你好,欢迎来到 Servlet 世界");
            out.println(" </body>");
            out.println("</html>");
            out.close();
        }
        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {

            doGet(request,response);
        }
    }
```

在示例 7-1 中,需要强调以下 3 点:

- 在调用 Servlet 时,首先要在程序中导入 Servlet 所需的包。
- 创建用于 Web 应用的 Servlet 继承自 HttpServlet 类。
- 实现 doGet()或者 doPost()方法。

那么如何访问该 Servlet 呢? 我们还需在 web.xml 中配置,配置如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.HelloServletTest</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/testServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

在上述代码中,首先通过<servlet-name>和<servlet-class>元素声明 Servlet 的名称和类的路径,然后通过<url-pattern>元素声明访问这个 Servlet 的 URI 映射。

打开 IE 浏览器,在地址栏中输入:http://localhost:8080/Ch07_1/testServlet,则会出现如图 7-4 所示的运行结果。



图 7-4 示例 7-1 的运行效果图

7.3 Servlet 与 JSP 的关系

既然 Servlet 与 JSP 都可以在页面上动态显示数据，那么它们之间存在什么样的关系？创建 Web 项目 Ch07_2 建立 Test.jsp 文件，内容如示例 7-2 所示。

示例7-2

```
/**
 * 简单的 Html 页面代码
 */
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Test JSP</title>
  </head>
  <body>
    This is my JSP page. <br>
  </body>
</html>
```

当我们部署项目并运行 Test.jsp 后，在 Tomcat 的安装目录下的

```
\work\Catalina\localhost\Ch07_2\org\apach\jsp
```

自动生成一个 Test_jsp.java，主要的内容如示例 7-3 所示。

示例7-3

```
/**
 * 部署后 work 目录下的 Test_jsp.java 文件
 */
……//导入需要引入的包
public final class Test_jsp extends
org.apache.jasper.runtime.HttpJspBase …… {
  public void _jspService(final
javax.servlet.http.HttpServletRequest request,
final javax.servlet.http.HttpServletResponse response)
        throws java.io.IOException, ServletException {
……//定义其他变量
    try {
      response.setContentType("text/html;charset=UTF-8");
      JspWriter out = pageContext.getOut();
      out.write("\r\n");
      out.write("<html>\r\n");
      out.write(" <head>\r\n");
      out.write(" <title>Test JSP</title>\r\n");
```

```
            out.write(" </head> \r\n");
            out.write(" <body>\r\n");
            out.write(" This is my JSP page. <br>\r\n");
            out.write(" </body>\r\n");
            out.write("</html>\r\n");
        } catch (java.lang.Throwable t) {
            ……//进行其他处理
        }
    }
```

从示例 7-3 中可以看出 Test.jsp 在运行时首先解析成一个 Java 类 Test_jsp.java，该类继承于 org.apache.jasper.runtime.HttpJspBase 类，而 HttpJspBase 又是继承自 HttpServlet 的类，由此我们可以得出一个结论，就是 JSP 在运行时会被 Web 容器翻译为一个 Servlet。

比较

JSP、JavaBean 和 Servlet，它们之间有什么区别和联系呢？

- Servlet 和 JavaBean 一样，本质都是 Java 类，不同的是，JavaBean 不能独立运行，只是提供接口供 JSP 等访问，而 Servlet 可以独立运行。
- 可以说 Servlet 是 JSP 的前身，在 JSP 出现之前，Sun 公司推出了 Servlet，但由于使用 Servlet 编写 HTML 脚本时，需要使用 print 或者 println 方法逐句打印输出，这给开发人员带来很大麻烦，限制了 Servlet 的广泛应用，由此，JSP 技术应运而生。JSP 网页是在 HTML 脚本中嵌入 Java 代码，它从根本上改变了 Servlet 的编程方式。使用 Servlet 显示页面的用户也越来越少。
- 就业务处理能力来说，Servlet 不如 JavaBean 和 EJB 强大。就页面显示能力来说，Servlet 不如 JSP 方便。既然这样，那 Servlet 岂不是没有什么用处？不是的！在当今的 J2EE 应用开发中，Servlet 仍然有它的用处，如处理小型的任务，或者用来作为 MVC（Model-View-Controller）模式中的控制器（Controller），下一章我们将介绍 MVC。使用 Servlet 作为控制器，控制模型（Model）和视图（View）之间的交互过程，它决定向用户返回怎样的视图等。因此 Servlet 的角色发生了改变。
- 选择 JSP 还是 Servlet，往往不是绝对的。常见的是将两者结合起来，例如，使用 Servlet 来处理用户请求，处理完毕，将结果发送给 JSP，由 JSP 来进行显示等。
- JSP、JavaBean 和 Servlet 可以进行交流，例如，JSP 可以调用 JavaBean，也可以调用 Servlet，在 Servlet 中处理数据后，也可以通过 JSP 网页显示出来等。

7.4 Servlet 的生命周期

为了在应用程序中能够更好地使用 Servlet，接下来我们了解一下 Servlet 的生命周期。所谓的生命周期就是 Servlet 从创建到销毁的过程，包括如何加载和实例化、初始化、处理请求以及如何被销毁。

1. 加载和实例化

Servlet 容器负责加载和实例化 Servlet，当客户端发送一个请求时，Servlet 容器会查找内存中是否存在该 Servlet 的实例，如果不存在，就创建一个 Servlet 实例。如果存在该 Servlet 的实例，就直接从内存中取出该实例来响应请求。

2. 初始化

在 Servlet 容器完成 Servlet 实例化后，Servlet 容器将调用 Servlet 的 init ()方法进行初始化，初始化的目的是让 Servlet 对象在处理客户端请求前完成一些初始化工作，例如，设置数据库连接参数，建立 JDBC 连接，或者是建立对其他资源的引用。init ()方法在 javax. servlet. Servlet 接口中定义。对于每一个 Servlet 实例，init()方法只被调用一次。

3. 服务

Servlet 被初始化以后，就处于能响应请求的就绪状态。当 Servlet 容器接收到客户端请求时，调用 Servlet 的 service()方法处理客户端请求。Servlet 实例通过 ServletRequest 对象获得客户端的请求。通过调用 ServletResponse 对象的方法设置响应信息。

4. 销毁

Servlet 的实例是由 Servlet 容器创建的，所以实例的销毁也是由容器来完成的。Servlet 容器判断一个 Servlet 是否应当被释放时(容器关闭或需要回收资源)，容器就会调用 Servlet 的 destroy ()方法，destroy ()方法指明哪些资源可以被系统回收，而不是 destroy ()方法直接进行回收。

Servlet 生命周期过程和相应的方法如图 7-5 所示：

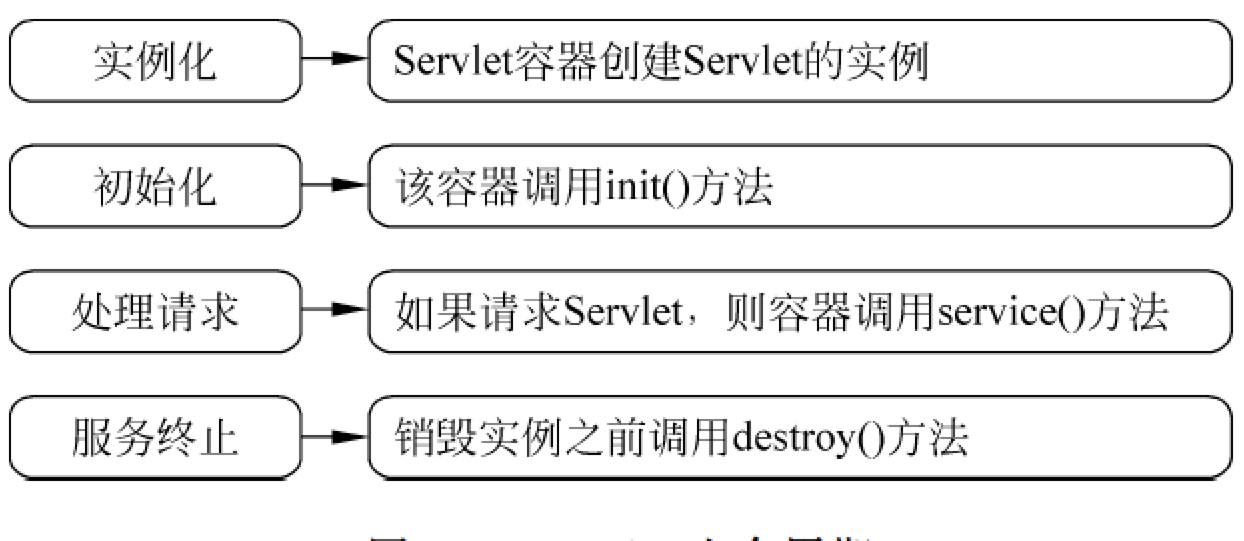


图 7-5　Servlet 生命周期

7.5　Servlet API

使用 Servlet API 可以开发 HTTP Servlet 或其他 Servlet，Servlet API 包含在两个包内。javax. servlet 包中的类和接口支持通用的不依赖协议的 Servlet，包括 Servlet、ServletRequest、ServletResponse、ServletConfig、ServletContext 接口及抽象类 GenericServlet；javax. servlet. http 包中的类和接口是用于支持 HTTP 协议的 Servlet API。

7.5.1　Servlet 接口

Servlet 接口定义了所有 Servlet 需要实现的方法，包括 init ()、service()、destroy ()方法，以及 getServletInfo ()方法(用于获得 Servlet 信息)和 getServletConfig ()方法(返回 ServletConfig 对象)。

7.5.2 GenericServlet 抽象类

抽象类 GenericServlet 实现了 Servlet 接口和 ServletConfig 接口，给出除 service()外的其他方法的简单实现，它定义了通用的、不依赖于协议的 Servlet。它常用的方法如表 7-1 所示。

表 7-1 GenericServlet 的常用方法

方 法 名 称	功 能 描 述
void init(ServletConfig config)	调用 Servlet 接口中的 init()方法
String getInitParameter(String name)	返回名称为 name 的初始化参数
ServletContext getServletContext()	返回 ServletContext 对象的引用

经验

通常只需要重写不带参数的 init()方法，如果重写 init(ServletConfig config)方法，那么应该包含 super.init(config)这句代码。

如果要编写一个通用的 Servlet，只要继承自 GenericServlet 类，实现 service()方法即可。

7.5.3 HttpServlet 抽象类

抽象类 HttpServlet 继承自 GenericServlet 类，具有与 GenericServlet 类似的方法和对象，支持 HTTP 的 post 和 get 方法，并提供了与 HTTP 相关的实现。HttpServlet 能够根据客户发出的 HTTP 请求，进行相应的处理，并得到相应的结果，然后这个相应结果会被自动封装到 HttpServletRequest 对象中。根据 HTTP 协议中定义的请求方法，HttpServlet 分别提供了处理请求的相应方法，如表 7-2 所示。

表 7-2 HttpServlet 的常用方法

方 法 名 称	功 能 描 述
void service(ServletRequest req, ServletResponse res)	调用 GenericServlet 类中 service()方法的实现
void doXXX(HttpServletRequest req, HttpServletResponse res)	根据请求方式的不同，分别调用相应的处理方法，例如，doGet()、doPost()等

经验

HttpServlet 的 service()方法，已经帮助我们根据请求方法的类型，调用相应的 doXxx()方法。所以在编写 Servlet 时只需要根据应用的需要，重写 doGet()或者 doPost()方法即可。

7.5.4 ServletConfig 接口

在 Servlet 初始化时 Servlet 容器使用 ServletConfig 对象向该 Servlet 传递信息。ServletConfig 接口定义的方法如表 7-3 所示。

表 7-3 ServletConfig 的常用方法

方法名称	功能描述
String getInitParameter(String name)	获取 web. xml 设置的以 name 命名的初始化参数值
ServletContext getServletContext()	返回 Servlet 的上下文对象引用



一个 Servlet 只有一个 ServletConfig 对象。

7.5.5 ServletContext 接口

一个 ServletContext 对象表示一个 Web 应用的上下文，Servlet 使用 ServletContext 接口定义的方法与它的 Servlet 容器进行通信。

Servlet 容器厂商负责提供 ServletContext 接口的实现，容器在应用程序加载时创建 ServletContext 对象，ServletContext 对象被 Servle 容器中的所有 Servlet 共享。

前面我们学习过的 JSP 内置对象 application 是 ServletContext 的实例，ServletContext 对象的用法详见表 7-4 所示。

表 7-4 ServletContext 的常用方法

方法名称	功能描述
String getInitParameter(String name)	获取名称为 name 的系统范围内的初始化参数值，系统范围内的初始化参数可以在部署描述符中使用<context-param>元素定义
void setAttribute(java. lang. String name, java. lang. Object object)	获取名称为 name 的属性
Object getAttribute(String name)	获取名称为 name 的属性
String getRealPath(String path)	返回相对路径的真实路径
void log(String message)	记录一般日志信息

注：ServletContext 还提供了很多实用的方法来取得服务器的信息，想进一步了解请参考 Servlet 文档。

7.5.6 ServletRequest 和 HttpServletRequest 接口

1. ServletRequest 接口

当客户请求时，由 Servlet 容器创建 ServletRequest 对象(用于封装客户的请求信息)这个对象将被容器作为 service ()方法的参数之一传递给 ServletRequest 对象获取客户端的请求数据。ServletRequest 接口中的常用方法如表 7-5 所示。

表 7-5 ServletRequest 接口的常用方法

方法名称	功能描述
Object getAttribute(String name)	获取名称为 name 的属性值
void setAttribute(String name,Object object)	在请求中保存名称为 name 的属性
void removeAttribute(String name)	清除请求中名字为 name 的属性

2. HttpServletRequest 接口

HttpServletRequest 位于 javax. servlet. http 包中，继承自 ServletRequest 接口。通过该接口同样可以获取请求中的参数 HttpServletRequest 接口除了继承了 ServletRequest 接口中的方法，还增加了一些用于读取请求信息的方法，增加的方法如表 7-6 所示。

表 7-6　HttpServletRequest 接口的常用方法

方 法 名 称	功 能 描 述
String getContextPath()	返回请求 URI 中表示请求上下文的路径，上下文路径是请求 URI 的开始部分
Cookie[ ] getCookies()	返回客户端在此次请求中发送的所有 Cookie 对象
HttpSession getSession()	返回和此次请求相关联的 Session，如果没有给客户端分配 Session，则创建一个新的 Session
String getMethod()	返回此次请求所使用的 HTTP 方法的名字，如 GET、POST

7.5.7　ServletResponse 和 HttpServletResponse 接口

1. ServletResponset 接口

Servlet 容器在接收客户请求时，除了创建 ServletRequest 对象用于封装客户的请求信息外，还创建了一个 ServletResponse 对象，用来封装响应数据，并且同时将这两个对象作为参数传递给 Servlet。Servlet 利用 ServletRequest 对象获取客户端的请求数据，经过处理后由 ServletResponse 对象发送响应数据。ServletResponse 接口中的常用方法如表 7-7 所示。

表 7-7　ServletResponse 接口的常用方法

方 法 名 称	功 能 描 述
PrintWriter getWriter()	返回 PrintWrite 对象，用于向客户端发送文本
String getCharacterEncoding()	返回在响应中发送的正文所使用的字符编码
void setCharacterEncoding()	设置发送到客户端的响应的字符编码
void setContentType(String type)	设置发送到客户端的响应的内容类型，此时响应的状态属于尚未提交

2. HttpServletResponse 接口

与 HttpServletRequest 接口类似，HttpServletResponset 接口也继承自 ServletResponse 接口，用于对客户端的请求执行响应。它除了具有 ServletResponse 接口的常用方法外，还增加了新的方法，如表 7-8 所示。

表 7-8　HttpServletResponse 接口的常用方法

方 法 名 称	功 能 描 述
void addCookie(Cookie cookie)	增加一个 Cookie 到响应中，这个方法可多次调用，设置多个 Cookie
void addHeader(String name,String value)	将一个名称为 name，值为 value 的响应报头添加到响应中

续表

方 法 名 称	功 能 描 述
void sendRedirect(String location)	发送一个临时的重定向响应到客户端,以便客户端访问新的 URL。抛出一个 IOException
void encodeURL(String url)	使用 Ssession ID 对用于重定向的 URL 进行编码,以便用于 sendRedirect() 方法中

7.6 Servlet 应用

7.6.1 获取 HTML 表单信息

在前面的章节中,已经介绍了使用 JSP 技术来接受 HTML 表单信息。同样的,Servlet 也可以接收客户浏览器在 HTML 表单中填入的信息,从而实现客户与服务器之间的交互。下面来举一个示例,该示例由一个 HTML 网页和一个 Servlet 程序组成。用户在 HTML 网页的表单中输入用户信息,包括姓名、性别和 Email 地址,并提交表单,Servlet 程序会接收这些信息,然后打印输出到用户浏览器中。

示例7-4

information.html 页面代码如下。

```
/**
 * 简单的 Html 页面代码
 */
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/
html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>servlet 获取 HTML 表单信息 </title>
</head>
<body>
    <form name="form1" action="informationServlet" method="post">
        姓名:<input name="name" type="text"><br/>
        性别:<input name="sex" type="text"><br/>
        Email 地址:<input name="email" type="text""><br/>
        <input type="submit" value="提交"><input type="reset" value="重填">
    </form>
</body>
</html>
```

接下来编写 Servlet 程序 informationServlet.java。

```
/**
 * Java servlet *
```

```
 */
package com.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class informationServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");
        request.setCharacterEncoding("utf-8");
        String name = request.getParameter("name");
        String sex = request.getParameter("sex");
        String email = request.getParameter("email");
        out.println("<HTML>");
        out.println(" <HEAD><TITLE>获取 HTML 表单信息</TITLE></HEAD>");
        out.println(" <BODY>");
        out.print("姓名: " + name + "<br>");
        out.print("性别: " + sex + "<br>");
        out.print("Email: " + email + "<br>");
        out.println(" </BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
            this.doGet(request, response);
    }
}
```

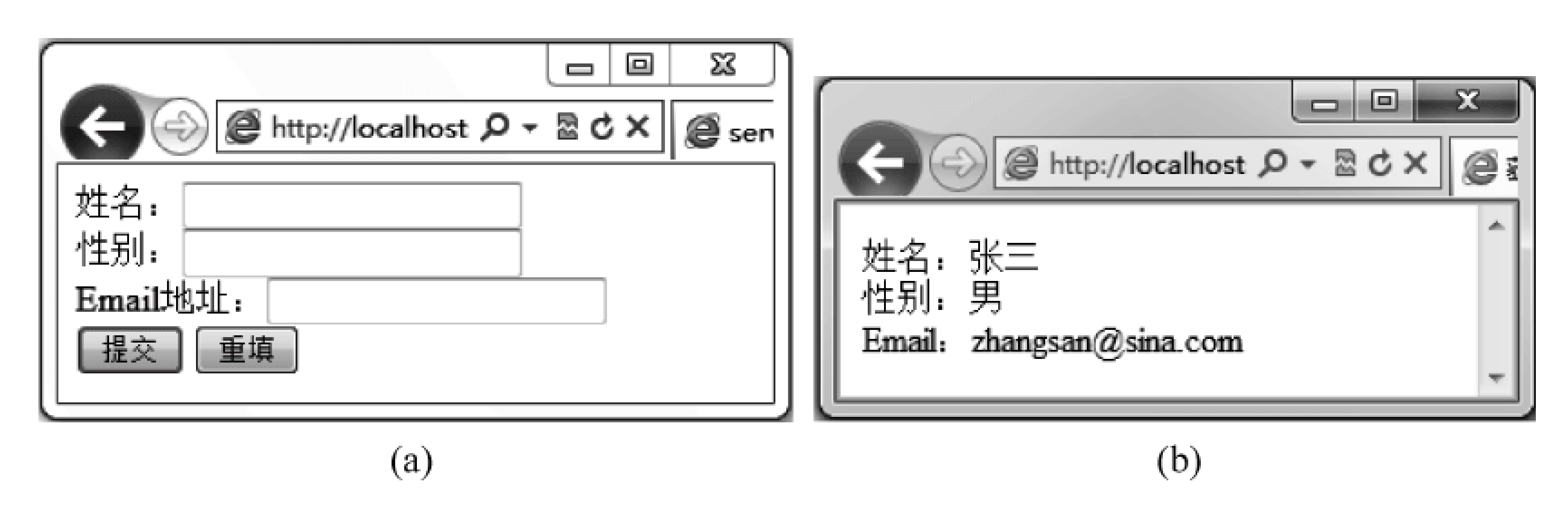
web.xml 中的配置为

```
/**
 * web.xml 的配置
 */
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet-name>informationServlet</servlet-name>
    <servlet-class>com.servlet.informationServlet</servlet-class>
```

```
    </servlet>
    <servlet-mapping>
      <servlet-name>informationServlet</servlet-name>
      <url-pattern>/informationServlet</url-pattern>
    </servlet-mapping>
  </web-app>
```

打开 IE 浏览器，在地址栏中输入：http://localhost:8080/Ch07_3/information.html，其运行结果如图 7-6(a)所示。

输入信息后单击“提交”按钮，运行结果如图 7-6(b)所示。



(a) (b)

图 7-6　示例 7-4 效果图

7.6.2　Servlet“控制器”

在前面的例子中，Servlet 可以接收 HTML 页面传入的值，并作出响应。其实 Servlet 还可以接收 JSP 传入的信息，以及调用 JavaBean，再把结果反馈给客户端。下面这个例子即采用了 JSP＋Servlet＋JavaBean 的形式进行开发。其中 Servlet 就像一个控制器一样根据不同的内容返回不同的结果。

示例7-5

创建 web 项目 Ch07_4

login.jsp 页面代码如下。

```
/**
 * 登录页面代码 *
 */
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Test Servlet</title>
  </head>
  <body>
   <form name="form1" action="loginServlet" method="post">
        用户名：<input name="name" type="text">
        <br>
        密码：<input name="password" type="password">
```

```
            <br>
            <input type="submit" value="登录"><input type="reset" value="重填">
        </form>
    </body>
</html>
```

在 src 目录下创建包 com.biz(业务处理),com.servlet(放置 servlet)。在 com.biz 中创建 UserBiz.java。

```
/**
 * 登录业务处理的 JavaBean *
 */
package com.biz;
public class UserBiz {
    public boolean login(String name,String password){
        if(name.equals("admin")&&password.equals("admin"))
            return true;
        else return false;
    }
}
```

此处简单处理：当用户名和密码为 admin 时,返回 true,否则返回 false(当然可以完善本例连接数据库判断)。

在 com.servlet 中创建 loginServlet.java。

```
/**
 * 处理登录的 servlet *
 */
package com.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.biz.UserBiz;
public class loginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        request.setCharacterEncoding("utf-8");
        String name=request.getParameter("name");
        String password=request.getParameter("password");
        UserBiz ub = new UserBiz();
        Boolean is = ub.login(name, password);
        if(is){
            //如果登录成功,则转发到 success.jsp 页面,并在页面中显示用户名
```

```
            request.setAttribute("loginName", name);
    request.getRequestDispatcher("success.jsp").forward(request, response);
        }else{
            //如果不符合要求,则重定向到登录页面
            response.sendRedirect("login.jsp");
        }
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

此 servlet 接收了 login.jsp 页面提交的 name 和 password,调用业务处理类 UserBiz.java 的 login(String name,String password)方法进行判断,再根据返回的结果确定是登录成功转发到成功页面,还是登录失败回到登录页面。登录成功后,还把用户名通过 request.setAttribute("loginName", name)的方法,放到 request 范围中,以便到 success.jsp 页面获取到用户名。

```
/**
 * 登录成功页面
 */
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登录成功页面</title>
  </head>

  <body>
      欢迎 <% =request.getAttribute("loginName") %>登录成功! <br>
  </body>
</html>
```

部署后,打开 IE 浏览器,在地址栏中输入:http://localhost:8080/Ch07_4/login.jsp,其运行结果分别如图 7-7(a)、(b)、(c)所示。

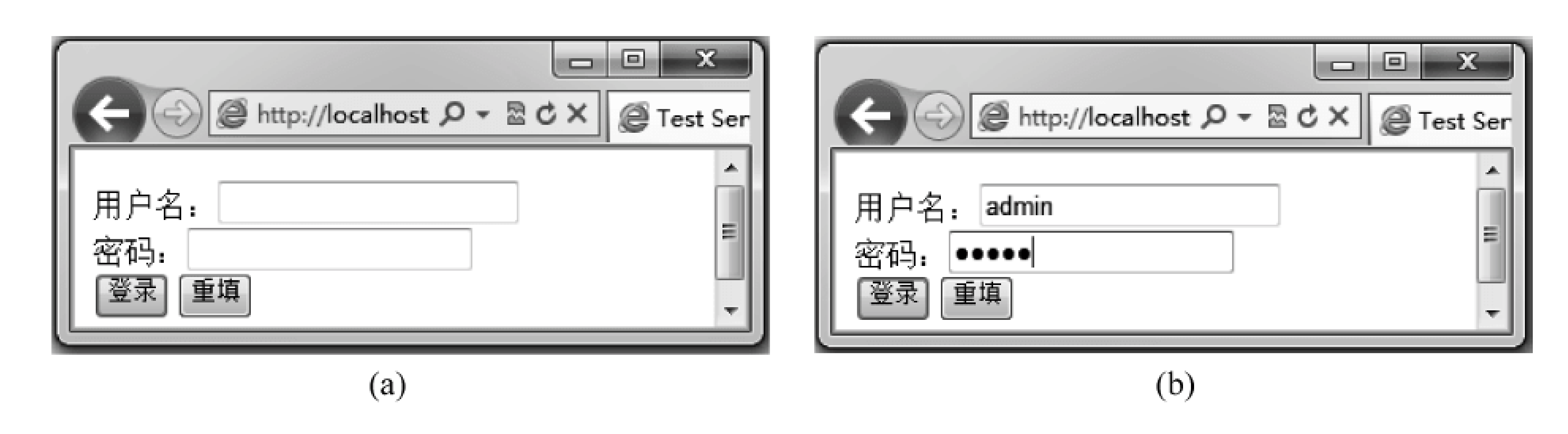


图 7-7 示例 7-5 效果图



(c)

图 7-7(续)

7.7 上 机 练 习

1. JSP+JavaBean+servlet 开发。

需求说明：当访问通知公告发布系统网站时，欢迎页面为前台页面。完善第 6 章上机练习 4、5、6，使用 JSP 调用 Servlet，Servlet 调用业务逻辑的 Java Bean 的方式完成。通知公告发布系统的前台显示通知公告类型列表、显示某一类型的通知公告列表、显示某一个通知的具体内容。

实现思路：在 Web. xml 中设置欢迎页面为 WebRoot 下的 index. jsp 页面。在第 6 章上机练习 4、5、6 的代码中，业务逻辑放在 JavaBean 中，JSP 调用业务逻辑时，调用封装业务逻辑的 JavaBean，这个上机练习接着完善此内容，使用 JSP 调用 Servlet，Servlet 调用业务逻辑的 JavaBean 的方式完成具体功能。所以要编写 Servlet 文件以及修改 JSP 页面。

提示代码：

显示

(1) index. jsp 中直接调用 Servlet 显示通知公告类型

```
<jsp:forward page="/typeServlet" />
```

(2) 当然在 web. xml 中配置

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
<servlet>
    <servlet-name>typeServlet</servlet-name>
    <servlet-class>com.servlet.TypeServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>typeServlet</servlet-name>
    <url-pattern>/typeServlet</url-pattern>
  </servlet-mapping>
```

(3) TypeServlet. java

```
package com.servlet;
import ……
public class TypeServlet extends HttpServlet {
```

```
    TypeBiz tb = new TypeBiz();
     public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        List list = tb.getAllType();
        request.setAttribute("list", list);request.getRequestDispatcher("page/foreground/
showIndex.jsp").forward(request,response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

其中 TypeBiz.java 为 type 的处理业务逻辑的 JavaBean。

（4）showIndex.jsp 显示类别名称的关键代码

```
<%      List TypeList = (List)request.getAttribute("list");
        if (TypeList != null && TypeList.size() != 0) {
              for (int i = 0; i < TypeList.size(); i++) {
              Type type = (Type) TypeList.get(i);
               %>
              <a href = "noticeServlet?method = showNotice&typeid = <% = type.getId() %>"
target = "showNotice">
              <% = type.getTypeName() %></a><br/>
        <%
              }
        }else{
              out.print("没有公告类别");
            }
 %>
```

（5）单击“通知公告类型”按钮，显示属于该类型的所有信息：调用 noticeServlet，并传入两个参数 method，typeid。target="showNotice"，指显示的内容在右侧的 showNotice 的 div 中。在 web.xml 中的配置为

```
<servlet>
  <servlet-name>noticeServlet</servlet-name>
  <servlet-class>com.servlet.NoticeServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>noticeServlet</servlet-name>
  <url-pattern>/noticeServlet</url-pattern>
</servlet-mapping>
```

（6）NoticeServlet. java

```
package com.servlet;
import ……
public class NoticeServlet extends HttpServlet {
    NoticeBiz nb = new NoticeBiz();
     public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html,charset=utf-8");
        response.setCharacterEncoding("utf-8");
        String method = request.getParameter("method");
        if("showNotice".equals(method))doShowNotice(request,response);
        else if("showNoticeDetail".equals(method))
        doShowNoticeDetail(request,response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        this.doGet(request, response);
    }
public void doShowNotice(HttpServletRequest request, HttpServletResponse response)  throws
ServletException, IOException {
int typeid = Integer.parseInt(request.getParameter("typeid"));
        List list = nb.getNoticeByType(typeid);
        request.setAttribute("list", list);request.getRequestDispatcher("page/foreground/
showNotice.jsp").forward(request,response);
    }
……
}
```

说 明

我们知道当 JSP 页面调用 Servlet 时，执行的是 doGet(或 doPost)方法，那么，当 JSP 页面要完成不同的功能又要调用同一个 Servlet 文件时，例如，页面要显示所有的 Notice 信息，以及显示某个 Notice 的详细信息时，doGet(或 doPost)方法中要同时写入完成这两个功能的代码，比较混乱。因此，可以由 JSP 页面中传入参数 method，Servlet 根据传入的参数确定用什么样的方法处理。

（7）showNotice. jsp 的关键代码为

```
<%
List NoticeList = new ArrayList(); //notice 列表
NoticeList = (List)request.getAttribute("list");
  if (NoticeList.size() != 0){
        %>
        <table border="0">
        <%
```

```
            for (int i = 0; i < NoticeList.size(); i++) {
                Notice notice = (Notice) NoticeList.get(i);
            %>
            <tr>
                <td><a href = "noticeServlet?method = showNoticeDetail&noticeId = <% = notice.
getId() %>" target = "showNotice"><% = notice.getTitle() %></a>
                </td>
            </tr>
                <%
            }
        }else{
            out.print("没有记录");
        } %>
```

接下来显示通知的详细信息，请仿照本例。

2. JSP+JavaBean+servlet 开发。

需求说明：通知公告发布系统的后台登录，使用 JSP 调用 Servlet，Servlet 调用业务逻辑的 JavaBean 的方式完成。

实现思路：页面内容提交到 servlet，servlet 调用业务逻辑，根据结果返回页面。

提示代码：

（1）login.jsp 的关键代码

```
<form action = "userServlet" method = "post" name = "form1" >
  用户名:<input type = "text" name = "loginName" size = "20"><br/>
  密   码:<input type = "password" name = "password" size = "20">
<input type = "submit" value = "登录" name = "submit">
  <input type = "reset" value = "重置" name = "reset">
  </p>
</form>
```

（2）web.xml 中配置

```
  <servlet>
    <servlet-name>userServlet</servlet-name>
    <servlet-class>com.servlet.UserServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>userServlet</servlet-name>
    <url-pattern>/userServlet</url-pattern>
  </servlet-mapping>
```

（3）UserServlet.java

```
package com.servlet;
import ……
```

```
public class UserServlet extends HttpServlet {
    UserBiz ub = new UserBiz();
     public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        request.setCharacterEncoding("utf - 8");
        String name = request.getParameter("loginName");
        String password = request.getParameter("password");
        User user = ub.login(name, password);
        if(user!= null){
    request.getSession().setAttribute("LOGINED_USER",user);
     request. getRequestDispatcher ( "/page/background/backIndex. jsp"). forward ( request,
response);
        }else{
            String message = "用户名密码错误,请重新登录!";
            request.setAttribute("message", message);
    request.getRequestDispatcher("login.jsp").forward(request,response);
        }
    }
    public void doPost (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        this.doGet(request, response);
    }
```

3. JSP＋JavaBean＋servlet 开发。

需求说明：通知公告发布系统的后台实现添加通知功能，使用 JSP 调用 Servlet，Servlet 调用业务逻辑的 JavaBean 的方式完成。

实现思路：页面内容通过 form 提交到 servlet，servlet 调用业务逻辑，根据结果返回页面。参考上机练习 2。

7.8 总　结

(1) Servlet 是一个符合特定规范的 Java 程序，在服务器端运行，处理客户端请求响应。

(2) Servlet 生命周期就是 Servlet 从创建到销毁的过程，包括如何加载和实例化、初始化、处理请求以及如何被销毁。

(3) 使用 Servlet API 可以开发 HTTP Servlet 或其他 Servlet。

7.9 作　业

一、选择题

1. 给定某 Servlet 的代码如下所示，运行该 Servlet 的结果是(　　)。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.PrintWriter;
```

```
import java.io.IOException;
public class ServletTest extends HttpServlet {
    public void init() throws ServletException{ }
    public void service(HttpServletReguest reg, HttpServletResponse res) throws ServletException,
IOException{
        PrintWriter out = response.getWriter();
        out.println("hello! ") ;
}
```

A. 编译不能够成功通过,提示缺少 doGet ()或者 doPost ()方法

B. 在浏览器中会看到输出文字 hello!

C. 在浏览器中看不到任何输出的文字

D. 在浏览器中会看到运行期错误信息

2. 以下对于 Servlet 描述错误的是(　　)。

A. Servlet 的生命周期:加载类和实例化、初始化、服务、销毁

B. Servlet 是基于 Java 技术的 Web 组件

C. Servlet 是由程序员自己编程实现的

D. Servlet 的 destroy()方法直接释放和回收资源

3. 当访问一个 Servlet 时,以下 Servlet 中(　　)方法被先执行。

A. destroy ()　　B. doGet()　　C. service()　　D. init()

4. 假设在一个名为 test 的 web 应用中有一个 MyServlet 类,在 web. xml 中对其进行如下的配置:

```
<servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>com.servlet.MyServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myServlet</url-pattern>
  </servlet-mapping>
```

则以下选项可以访问到 MyServlet 的是(　　)。

A. http://localhost:8080/MyServlet

B. http://localhost:8080/myServlet

C. http://localhost:8080/com/servlet/MyServlet

D. http://localhost:8080/test/myServlet

二、简答题

1. 什么是 Servlet? Servlet 与 JSP 有什么区别?

2. 运行 Servlet 需要在 web. xml 中如何配置?

3. Servlet 生命周期是什么?

三、程序题

1. 创建一个 servlet,要求通过浏览器地址栏中访问该 servlet 后,输出一个一行一列的

表格,单元格里的内容是“节约光荣,浪费可耻”。

2. 实现一个简单的登录程序。要求由 servlet 接收用户输入的用户名和密码,然后输出到页面中。

第 8 章　MVC 设计模式

本章学习目标

- 了解什么是 Model Ⅰ模式
- 了解 Model Ⅰ两种开发方式
- 了解 Model Ⅰ模式的优缺点
- 了解什么是 Model Ⅱ模式
- 了解 Model Ⅱ模式的优缺点
- 掌握什么是 MVC 模式
- 了解 MVC 模式的优缺点
- 使用 MVC 模式开发应用程序

通过学习我们知道,JSP 技术的主要任务是简化页面的开发。在编写程序的时候,若把大量的业务逻辑代码写在 JSP 页面中,进行程序控制和业务逻辑的操作,则这违背了 JSP 技术的初衷,为程序员和美工带来了很大的困扰,为了解决这些问题,在应用程序设计时广泛采用了 MVC 设计模式。

8.1　Model Ⅰ和 Model Ⅱ

在学习 MVC 之前,首先要了解两种架构模式:Model Ⅰ和 Model Ⅱ。

在没有学习 Servlet 技术时,采用 JavaBean 封装数据和实现业务逻辑,程序的控制由 JSP 完成,这种 JSP+JavaBean 的做法是早期使用 JSP 开发 Web 应用常用的方法。JSP 实现页面的显示,JavaBean 对象用来封装数据和实现业务逻辑,这种方法被称为 Model Ⅰ模式。

8.1.1　Model Ⅰ模式

Model Ⅰ模式有两种开发方式:一种是纯 JSP 方式的开发,另一种是使用 JSP + JavaBean 的开发。

1. 纯 JSP 方式开发

最直观的使用 JSP 开发 Web 应用的方法是在 JSP 中直接嵌入 Java 代码,即小脚本方式,所有的逻辑控制和业务处理都以小脚本的方式实现。使用这种方法的优点是简单方便,

适合搭建小型的 Web 应用，但这样会使页面显得非常混乱，并且不易于后期的维护扩展。使用纯 JSP 方式开发的结构图如图 8-1 所示。

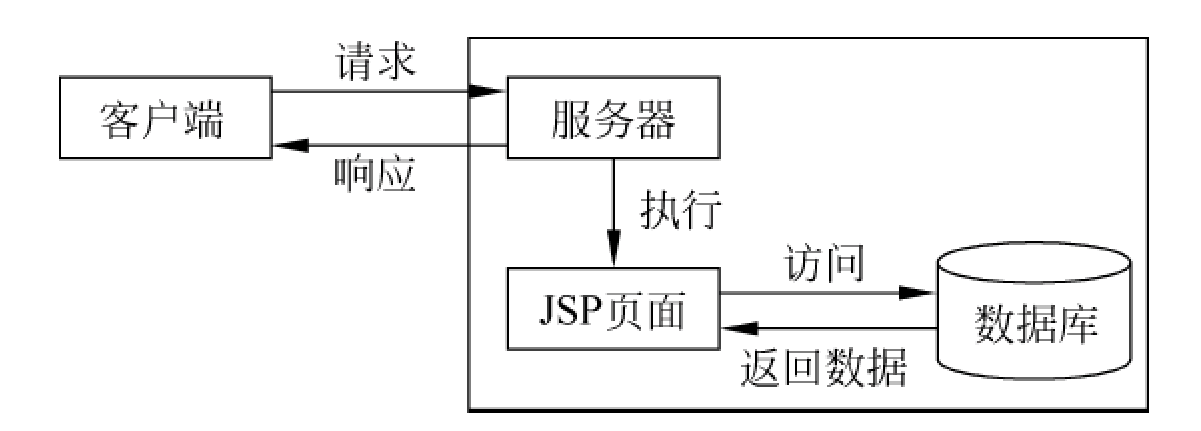
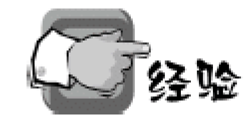


图 8-1 纯 JSP 方式开发结构图



这种纯 JSP 方式开发只适用于刚接触 JSP 时使用，在实际开发中不推荐使用。

2. JSP＋JavaBean 方式开发

对纯 JSP 方式开发作出一些改进，使用 JavaBean 封装数据和进行业务处理后，使得页面简洁有利于代码的重用和后期维护，但还是存在很多的限制，因为程序的逻辑控制还是由 JSP 完成，页面中依然需要嵌入大量的 Java 代码。使用 JSP ＋ JavaBean 方式开发的结构图如图 8-2 所示。

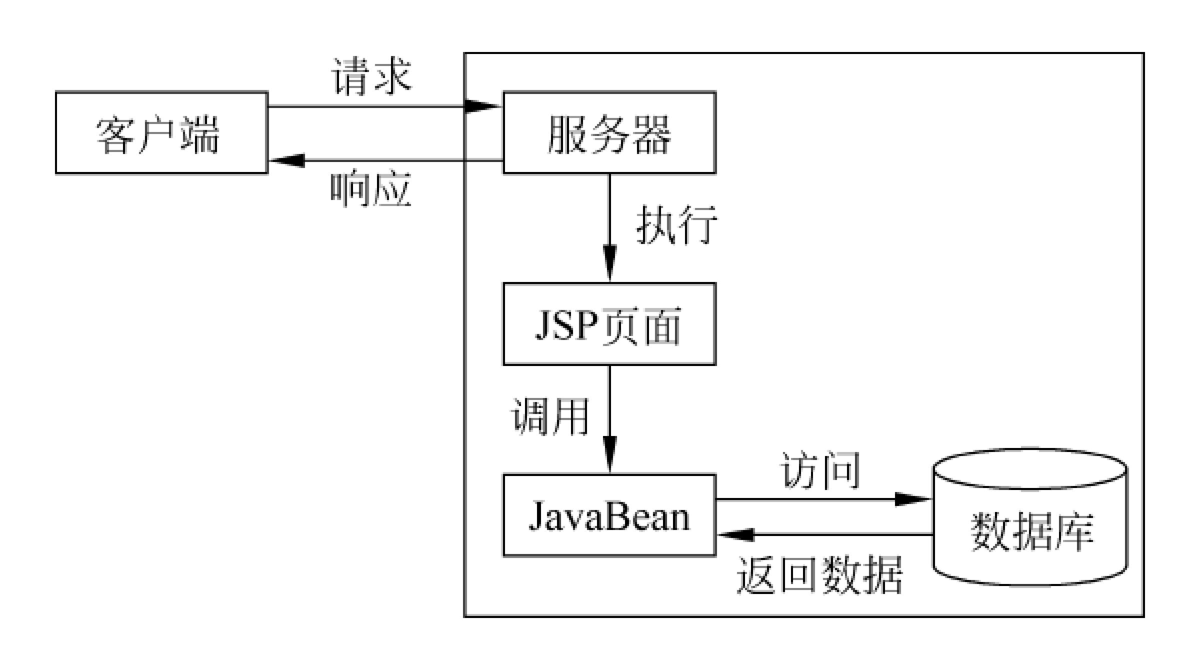


图 8-2 使用 JSP＋JavaBean 方式开发的结构图

不论哪种开发方式，Model Ⅰ模式的缺点非常明显。

1. 不利于后期维护与扩展

Model Ⅰ模式由于逻辑控制代码与显示代码是混在一起的，会导致页面设计与逻辑控制无法分离，使程序可读性差，调试困难，功能划分不清，不利于维护与项目的扩展。

2. 不利于页面维护

当构建一个项目的时候，必须考虑到美工美化界面的问题。JSP 与 Java 代码混合交织在一起，美工就会一头雾水。

8.1.2 ModelⅡ模式

由于 Model Ⅰ模式存在不足，当程序流程非常复杂的时候，要修改一个程序带来的工作量将非常大。为了克服 Model Ⅰ模式的缺陷，人们引入了 Model Ⅱ模式。在 Model Ⅰ模式中 JSP 页面嵌入了流程控制代码和业务逻辑处理代码，将这部分代码提取出来，放入单独的类(Servlet 和 JavaBean)中，也就是使用 JSP＋Servlet ＋JavaBean 共同开发应用程序，

这种方式就是 Model Ⅱ模式。

Model Ⅱ架构模式的工作原理如图 8-3 所示。

如图 8-3 所示，Model Ⅱ架构模式的工作流程是按照如下 5 个步骤进行的：

1. Servlet 接收客户端发出的请求；
2. Servlet 根据不同的请求调用相应的 JavaBean；
3. 业务逻辑调用数据访问的 JavaBean 访问数据库，返回结果；
4. Servlet 将接收 JavaBean 的结果传递给 JSP 视图；
5. JSP 将后台处理结果呈现给客户端。

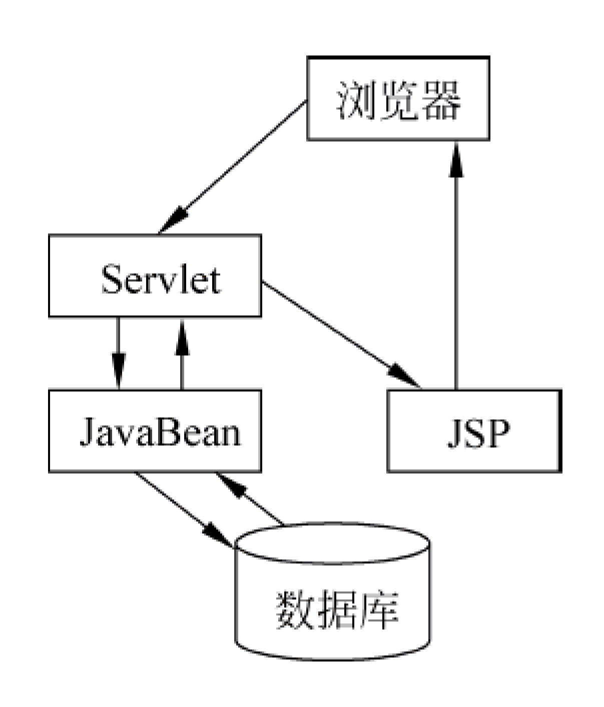


图 8-3　Model Ⅱ架构模式的工作原理图

Model Ⅱ模式体现了基于 MVC（Model-View-Controller，模型-视图-控制器）的设计模式，简单地说，就是将数据显示、流程控制和业务逻辑处理分离，使之相互独立。

8.2　MVC 模式

设计模式是一套被反复使用、成功的代码设计经验的总结。模式必须是典型问题（不是个别问题）的解决方案。

8.2.1　MVC 设计模式

在程序设计中，把采用模型（Model）、视图（View）、控制器（Controller）的设计方式称为 MVC 设计模式。MVC 是一种流行的软件设计模式，代表了一种多层的应用程序实现方式。MVC 模式将应用程序实现分为三个不同的基本部分。

- 模型（Model）：用来表示数据和处理业务。对应的组件是 JavaBean。模型返回的数据是中立的，即模型与数据格式无关，一个模型能为多个视图提供数据。提高了代码的重用性。模型分为业务模型和数据模型。
- 视图（View）：是用户看到并与之交互的界面。对应的组件是 JSP 或 HTML 文件。视图提供可交互的客户界面，向客户显示模型数据。在视图中其实没有真正的处理发生，不管这些数据是什么，作为视图来讲它只是作为一种输出数据并允许用户操纵的方式。
- 控制器（Controller）：接受用户的输入并调用模型和视图去完成用户的请求。对应的组件是 Servlet。控制器响应客户的请求，根据客户的请求来操作模型，并把模型的响应结果经由视图展现给客户。当用户提交一个请求时，控制器本身不输出任何东西，不做任何业务处理。它只要接收请求并决定调用哪个模型组件去处理请求，然后确定用哪个视图来显示模型处理返回的数据。

8.2.2　MVC 模式的编程思路

在使用 MVC 模式进行编程时，要注意各个组件的分工与协作。基于 MVC 模式的 Web 应用的基本工作流程可以分为 4 个步骤（如图 8-4 所示）：

1. 用户通过视图发出请求；

2. 控制器接收请求后，调用相应的模型来处理具体的业务；
3. 控制器根据返回的结果，选择对应的视图组件来反馈结果；
4. 视图根据接收到的结果，将信息显示给用户。

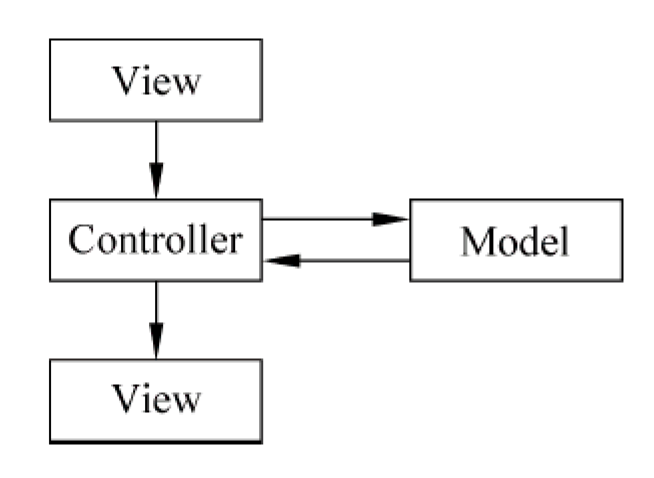


图 8-4 MVC 模式工作流程

8.2.3 MVC 模式的优缺点

虽然 MVC 设计模式在面向对象程序设计中被广泛应用，但是该设计模式并不是十全十美的。我们必须了解并掌握 MVC 的优点及缺点，这样在实际开发过程中才能够扬长避短，充分发挥 MVC 设计模式的优势。

1. MVC 的优点体现在如下 3 个方面

- 有利于分工部署。使用 MVC 设计模式开发应用程序，不同的人员分工非常明确。Java 程序员只需将精力集中于业务逻辑，而界面程序员（HTML 和 JSP 开发人员）只需将精力集中页面表现形式上。
- 降低耦合，提高可维护性。MVC 设计模式将表示层与业务层有效分离，降低了二者之间的耦合紧密程度。这样一来任何业务逻辑的变动都不会影响到表示层代码；同样开发人员可以随意修改表示层代码，而不用重新编译模型和控制器的代码。
- 提高应用程序的重用性。对于同一个 Web 应用，客户可能会使用多种方式进行访问，可以通过计算机的 HTML 浏览器进行访问，也可以通过移动电话的 WAP 浏览器进行访问。但是无论通过什么访问方式，应用程序的业务逻辑以及业务流程都是一样的，需要改变的只是表示层的具体实现方式。由于 MVC 实现了业务与视图的分离，所以能够轻松解决这一问题。

2. MVC 的缺点体现在如下 2 个方面

- MVC 并不适合小型应用程序的开发设计。MVC 要求开发人员完全按照模型、视图及控制器 3 个组件的模式对应用程序进行划分。对于某些小型应用来说，反而会增加一些不必要的工作，影响开发效率。
- 基于 MVC 设计模式进行程序设计，要求开发人员在设计编程之前，必须精心设计程序结构；在设计过程中，由于将一个完整的应用划分为 3 个组件，相应增加了需要管理文件的数量。

8.3 开发基于 MVC 模式的应用程序

其实在第 7 章上机练习中我们已经使用了 Model Ⅱ 模式。现在回想一下，我们是如何实现登录的，是不是已经使用了 Model Ⅱ 模式呢？那么本节我们就按照 Model Ⅱ 模式来开

发通知公告发布系统的全部功能。

1. 视图开发

实现通知公告发布系统，分为前台和后台，首先创建 View 的 JSP 页面，整个项目的页面目录如图 8-5 所示，前台页面显示列表如图 8-6 所示。

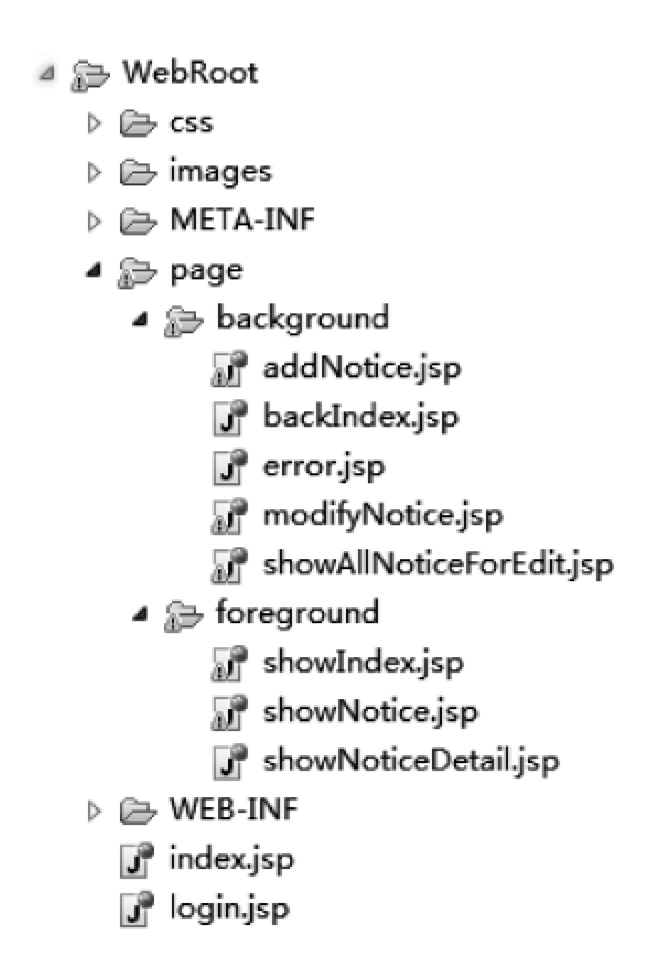


图 8-5　通知公告发布系统的作为 View 的 JSP 页面目录结构



图 8-6　通知公告发布系统——前台页面显示列表

index.jsp 进入后，右面是没有内容的，当单击左侧的通知公告类别下的名称，右侧才出现属于该类的所有通知公告信息的链接，当单击每个链接时，显示的每个通知公告的具体内容如图 8-7 所示。



图 8-7 通知公告发布系统——前台页面显示通知的具体信息

进入后台，首先是如图 8-8 所示的登录页面。

图 8-8 通知公告发布系统——后台登录页面

登录成功后显示如图 8-9 所示的页面(以上功能第 7 章上机练习均已实现)。

单击通知公告列表链接，显示如图 8-10 所示的页面，每个通知公告后有修改、删除操作链接。

单击“修改”操作链接，显示如图 8-11 所示的页面。

修改成功后的显示如图 8-10 所示，单击“删除”操作链接，成功后返回的页面如图 8-10 所示。



图 8-9 “通知公告发布系统——后台”登录成功页面



图 8-10 “通知公告发布系统——后台”显示通知公告列表页面



图 8-11 “通知公告发布系统——后台”修改页面

单击“添加通知公告”操作链接，显示如图 8-12 所示，添加成功后的显示如图 8-10 所示。



图 8-12 “通知公告发布系统——添加”通知公告页面

2. 模型开发

通知公告发布系统的模型部分，在前面已经陆陆续续地实现了，整个的模型的结构如图 8-13 所示。

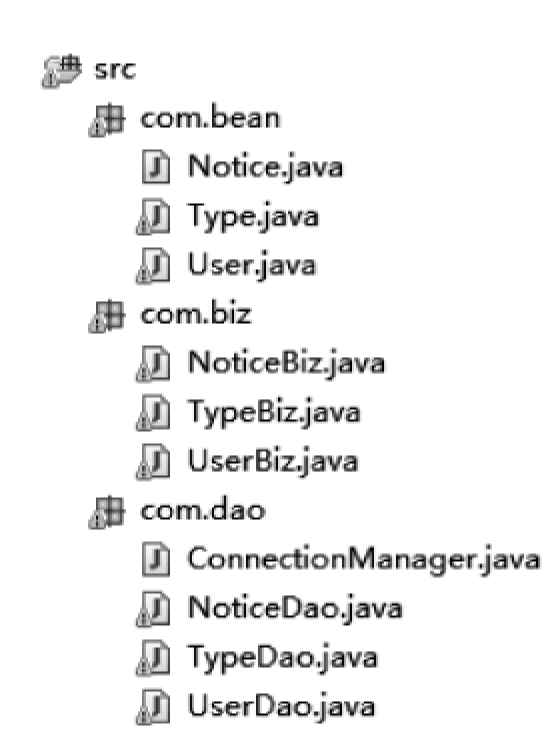


图 8-13　通知公告发布系统的作为 model 的结构图

在模型开发结构中分为三块，com. bean 包中存放实体类，com. dao 包中存放数据访问类，com. biz 存放业务逻辑类。本案例的业务逻辑简单，在 dao 和 biz 层没有出现接口，当开发的 Web 应用较复杂时，可以出现数据访问和业务逻辑的接口层，和实现类层。

3. 控制器开发

ModelⅡ模式使用 Servlet 作为 Controller，作用就是接收用户的请求数据，选择合适的 Model 处理具体的业务，处理完成后，根据 Model 返回的结果选择一个 View 显示数据。具体的结构图如图 8-14 所示。

com.servlet
NoticeServlet.java
TypeServlet.java
UserServlet.java

图 8-14　通知公告发布系统的作为 controller 的结构图

8.4 上机练习

本章上机练习，我们用 MVC 模式完成贯穿案例。通知公告发布系统的前台及登录功能在第 7 章已经实现，本章练习重点完成通知公告发布系统的后台页面。

1. 开发基于 MVC 模式的通知公告系统后台。

需求说明：在前几章的上机练习的基础上完善通知公告发布系统——后台，当单击“通知公告列表”操作链接时，在右面显示所有的通知公告标题并显示修改和删除链接。即编写 JSP 页面，完成图 8-10 的页面。

实现思路：在 JSP 页面 backIndex. jsp 中的通知公告列表处创建链接到 noticeServlet，并传入参数 method，完善 NoticeServlet. java，doGet 方法中接收 method 的值，经过判断后调用 doshowNotice 方法，用以显示所有的 Notice 信息到 showAllNoticeForEdit. jsp 页面。

参考代码如下：

（1）backIndex.jsp

```
… …
<div class = "notice_content">
        <a href = "noticeServlet?method = showAllNotice" target = "showNotice">通知公告列表
</a>
</div>
… …
```

（2）NoticeServlet.java

```
… …
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        … …
        String method = request.getParameter("method");
        … …
if("showAllNotice".equals(method))doShowAllNotice(request,response);
        … …
    }
public void doShowAllNotice(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        List list = nb.getAllNotice();
        request.setAttribute("list", list);
        request.getRequestDispatcher("page/background/showAllNoticeForEdit.jsp").forward
(request,response);
    }
… …
```

（3）showAllNoticeForEdit.jsp

```
… …
<%
    List NoticeList = (List) request.getAttribute("list");
    if (NoticeList.size() != 0){
  %>
    <table border = "0">
        <tr>
            <td>通知标题</td><td>操作</td>
        </tr>
            <%
                for (int i = 0; i < NoticeList.size(); i++) {
                Notice notice = (Notice) NoticeList.get(i);
                int noticeId = notice.getId();
            %>
        <tr>
            <td><a href = "noticeServlet?method = showNoticeDetail&noticeId = <% = noticeId %>"
target = "showNotice"><% = notice.getTitle() %></a>
```

```
                </td>
                <td><a
                href = "noticeServlet?method = showNoticeDetailForModify&noticeId = <% = noticeId %
>" target = "showNotice">修改</a></td>
                <td><a
                href = "noticeServlet?method = delectNotice&noticeId = <% = noticeId %>" target
 = "showNotice">删除</a></td>
            </tr>
                <%
            }
        }else{
            out.print("没有记录");
        }   %>
        </table>… …
```

2. 开发基于 MVC 模式的通知公告系统后台。

需求说明：接上机练习 1，完成修改功能。

实现思路：在 showAllNoticeForEdit.jsp 页面中，在标题后添加修改链接，链接到 noticeServlet 并传入参数 method，接着完善 NoticeServlet.java，doGet 方法中接收 method 的值，经过判断后调用 doshowNoticeDetailForModify 方法，用以显示要修改的 Notice 信息到 modifyNotice.jsp 页面；modifyNotice.jsp 页面用表单显示要修改的 notice 的信息，当修改完成后提交到 noticeServlet，并传入隐藏的参数 method，接着完善 NoticeServlet.java，doGet 方法中接收 method 的值，经过判断后调用 doModifyNotice 方法，接着返回到 showAllNoticeForEdit.jsp 页面。

（1）NoticeServlet.java

```
… …
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        … …
        String method = request.getParameter("method");
        … …
if ( " showNoticeDetailForModify ". equals ( method )) doShowNoticeDetailForModify ( request,
response);
        … …
    }
public void doShowNoticeDetailForModify (HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        int noticeId = Integer.parseInt(request.getParameter("noticeId"));
        Notice notice = nb.getNoticeById(noticeId);
        request.setAttribute("notice", notice);
        TypeBiz tb = new TypeBiz();
        List list = tb.getAllType();
        request.setAttribute("list", list);
```

```
        request.getRequestDispatcher("page/background/modifyNotice.jsp").forward(request,
    response);}
    ……
```

(2) modifyNotice.jsp

```
    ……
   <% Notice notice = (Notice)request.getAttribute("notice");
       List list = (List)request.getAttribute("list");
   %>
       <form method = "post" name = "form1" action = "noticeServlet">
       <table>
       <tr><td>标题:</td><td><input type = "text" name = "title" size = "35" value = "<% =
   notice.getTitle() %>"/></td></tr>
       <tr><td>作者:</td><td><input type = "text" name = "editor" size = "35" value = "<% =
   notice.getEditor() %>"/></td></tr>
       <tr><td>内容:</td><td><textarea name = "content" rows = "10" cols = "30" ><% =
   notice.getContent() %></textarea></td></tr>
       <tr><td> 类型:</td><td><select name = "type">
       <%
       for(int i = 0;i<list.size();i++){
       Type type = (Type)list.get(i);
           %>
               <option value = "<% = type.getId() %>"><% = type.getTypeName() %></option>
           <%
       }
        %>
           </select></td></tr>
       <tr><td><input type = "hidden" name = "method" value = "modifyNotice"/>
               <input type = "hidden" name = "id" value = "<% = notice.getId() %>"/>
               <input type = "submit" value = "提交" name = "submit" onclick = "return addSubmit
   ();"/></td>
           <td><input type = "reset" value = "重置" name = "reset"/>
               </td></tr>
       </table>
       </form>……
```

(3) NoticeServlet.java

```
    ……
   public void doGet(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, IOException {
           ……
           String method  =  request.getParameter("method");
           ……
   if("modifyNotice ".equals(method)) doModifyNotice(request,response);
           ……
       }
```

```
    public void doModifyNotice(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        String title = request.getParameter("title");
        String editor = request.getParameter("editor");
        String content = request.getParameter("content");
        int type = Integer.parseInt(request.getParameter("type"));
        int id = Integer.parseInt(request.getParameter("id"));
        Notice notice  = new Notice();
        notice.setId(id);
        notice.setContent(content);
        notice.setEditor(editor);
        notice.setTitle(title);
        notice.setType(type);
        Boolean is  =  nb.modifyNotice(notice);
        if(is){
            List list  =  nb.getAllNotice();
              request.setAttribute("list", list); request.getRequestDispatcher("page/
background/showAllNoticeForEdit.jsp").forward(request,response);
        }else{
             request.getRequestDispatcher("page/background/error.jsp").forward(request,
response);
      }
    }… …
```

3. 开发基于 MVC 模式的通知公告系统后台。

需求说明：接上机练习 1，完成删除功能。

实现思路：在 showAllNoticeForEdit.jsp 页面中，在标题后添加删除链接，链接到 noticeServlet 并传入参数 method，接着完善 NoticeServlet.java，doGet 方法中接收 method 的值，经过判断后调用 doDeleteNotice 方法，用以删除 Notice 信息，接着返回到 showAllNoticeForEdit.jsp 页面。

NoticeServlet.java

```
… …
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        … …
        String method  =  request.getParameter("method");
        … …
if("delectNotice ".equals(method)) doDelectNotice (request,response);
        … …
    }
public void doDeleteNotice (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        int noticeId  =  Integer.parseInt(request.getParameter("noticeId"));
        Boolean is  =  nb.delectNotice(noticeId);
        if(is){
            List list  =  nb.getAllNotice();
```

```
            request.setAttribute("list", list);
             request.getRequestDispatcher("page/background/showAllNoticeForEdit.jsp").
forward(request,response);
        }else{
             request.getRequestDispatcher("page/background/error.jsp").forward(request,
response);
        }……
```

4. 开发基于 MVC 模式的通知公告系统后台。

需求说明：完成添加通知公告功能。

实现思路：在 JSP 页面 backIndex.jsp 中的添加通知公告处创建链接到 noticeServlet，并传出参数 method，完善 TypeServlet.java，doGet 方法中接收 method 的值，经过判断后页面转发到 page/background/addNotice.jsp 页面；addNotice.jsp 页面创建表单，用以添加 Notice 信息，并且该表单提交到 noticeServlet，接着完善 NoticeServlet.java. doGet 方法中接收 method 的值，经过判断后调用 doAddNotice 方法，用以添加 Notice 信息，接着返回到 showAllNoticeForEdit.jsp 页面。

（1）backIndex.jsp

```
……
<div class="notice_content">
                    <a href="typeServlet?method=addNoticeForType" target="showNotice"
>添加通知公告</a>
</div>……
```

（2）TypeServlet.java

```
……
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String method=request.getParameter("method");
        List list = tb.getAllType();
        request.setAttribute("list", list);
    if(method.equals("addNoticeForType"))request.getRequestDispatcher("page/background/
addNotice.jsp").forward(request,response);
    else
if(method.equals("showType"))request.getRequestDispatcher("page/foreground/showIndex.
jsp").forward(request,response);
    }……
```

（3）addNotice.jsp

```
……
<%
    List list=(List)request.getAttribute("list");
%>
    <form method="post" name="form1" action="noticeServlet">
```

```
    <table>
    <tr><td>标题:</td><td><input type="text" name="title" size="35"><input type
="hidden" name="method" value="addNotice"></td></tr>
    <tr><td>作者:</td><td><input type="text" name="editor" size="35"></td></tr>
    <tr><td>内容:</td><td><textarea name="content" rows="10" cols="30"></textarea
></td></tr>
    <tr><td>类型:</td><td><select name="type">
    <%
    for(int i=0;i<list.size();i++){
    Type type=(Type)list.get(i);
     %>
    <option value="<% =type.getId() %>">
            <% =type.getTypeName() %></option>
        <%
        }
     %>
        </select></td></tr>
    <tr><td><input type="submit" value="提交" name="submit" onclick="return
addSubmit();"></td>
        <td><input type="reset" value="重置" name="reset"></td>
    </tr>
    </table>

    </form>……
```

(4) NoticeServlet.java

```
……
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ……
        String method = request.getParameter("method");
        ……
if("addNotice".equals(method)) doAddNotice (request,response);
        ……
    }
    public void doAddNotice (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        String title=request.getParameter("title");
        String editor=request.getParameter("editor");
        String content=request.getParameter("content");
        int type=Integer.parseInt(request.getParameter("type"));
        Notice notice = new Notice();
        notice.setContent(content);
        notice.setEditor(editor);
        notice.setTitle(title);
        notice.setType(type);
        Boolean is = nb.addNotice(notice);
```

```
            if(is){
                List list = nb.getAllNotice();
                  request.setAttribute("list", list); request.getRequestDispatcher("page/
background/showAllNoticeForEdit.jsp").forward(request,response);
            }else{
        request.getRequestDispatcher("page/background/error.jsp").forward(request,response);
            }
        }……
```

8.5 总　　结

(1) 设计模式是某一类问题的解决方案，是一套可以被反复使用，成功的代码设计经验的总结。

(2) Model Ⅰ模式有两种开发方式：一种是纯JSP方式的开发，另一种是使用JSP＋JavaBean的开发。

(3) 将JSP页面的流程控制代码和业务逻辑处理代码提取出来，放入单独的类(Servlet和JavaBean)中，也就是使用JSP＋Servlet＋JavaBean共同开发应用程序，这种方式就是Model Ⅱ模式。

(4) MVC模式将将系统分为三个不同的模块：

- 模型(Model)：对应的组件是JavaBean (Java类)；
- 视图(View)：对应的组件是JSP或HTML文件；
- 控制器(Controller)：对应的组件是Servlet。

(5) 由Servlet接收客户端请求，调用相应的模型处理业务逻辑和数据，再由Servlet根据处理结果，选择相应的JSP或HTML文件响应客户端。

(6) MVC的优点体现在如下3个方面：

- 有利于分工部署；
- 降低耦合，提高可维护性；
- 提高应用程序的重用性。

(7) MVC的缺点体现在如下2个方面：

- MVC并不适合小型应用程序的开发设计；
- 基于MVC设计模式进行程序设计，要求开发人员在设计编程之前，必须精心设计程序结构；在设计过程中，由于将一个完整的应用划分为3个组件，相应增加了需要管理文件的数量。

8.6 作　　业

一、选择题

1. 以下不属于MVC设计模式中三个模块的是(　　)。

A. 模型层　　B. 表示层　　C. 视图层　　D. 控制器

2. 使用 MVC 模式设计的 Web 应用程序具有以下优点，除了（　　）。

A. 可维护性强　　B. 可扩展性强　　C. 代码重复较少　　D. 大大减少代码量

3. 在 MVC 模式中（　　）专用于客户端应用程序的图形数据表示，与实际数据处理无关。

A. 模型　　B. 数据　　C. 视图　　D. 控制器

4. 在 MVC 设计模式中（　　）接收用户请求数据。

A. HTML　　B. JSP　　C. Servlet　　D. 业务类

二、简答题

1. MVC 的各个模块都是由哪些技术来实现的？

2. 使用 MVC 模式开发的优势是什么？缺点是什么？

3. 在程序开发的过程中，如何实现 MVC 模式？

三、程序题

编写基于 MVC 模式的程序，实现注册功能，注册后显示个人信息。

第 9 章　JSP 开发业务应用

本章学习目标

- 掌握分页显示的原理及实现步骤
- 掌握 SmartUpload 组件上传文件的功能
- 掌握 SmartUpload 组件实现文件下载的功能

通过前面的学习,我们已经基本掌握了 JSP 技术,包括 JSP 的组成、常用指令、内置对象、JDBC、JSP 处理客户端请求、servlet 技术等,通过这些知识能够开发出简单的 Web 应用程序。然而目前基于 Internet 的 Web 应用越来越丰富,数据量也越来越大,开发 Web 应用程序的难度也随之增加。

本章将要学习两种非常实用的技术:数据分页显示与文件上传、下载。

9.1　JSP 分页技术

随着时代的发展,基于 Internet 的 Web 应用也变得越来越复杂,资源也越来越庞大。以列表的形式显示数据,能够按照指定格式显示,使布局清晰,不受信息数量的限制。但是当数据量很大时,仍然以列表的形式显示,受页面的限制用户必须拖动页面才能浏览更多的数据,而且页面也显得冗长。那么,有没有一种显示方式,既能显示多条信息,又不需要拖动页面呢?答案就是以分页的形式显示数据。

以分页的形式显示数据,使数据更加清晰直观,页面不再冗长,也不受数据量的限制。目前有多种方式可以实现分页,其中一种是将所有查询结果保存在 session 对象或集合中,翻页的时候从 session 或集合中取出一页所需的数据显示。这种方法主要有两个的缺点:一是用户看到的可能是过期数据;二是如果数据量非常大,查询一次结果集会耗费很长时间,并且存储的数据也会占用大量内存,效率明显下降。其他常见的方法还有使用存储过程进行分页,但是需要设置主键和索引来提高查询效率,并且可移植性不高。

比较好的分页做法是每次翻页的时候只从数据库里检索出本页需要的数据。虽然每次翻页都查询数据库,但查询出的记录数很少,网络传输量不大。而在数据库端有各种成熟的技术用于提高查询速度,比在数据库的客户端保存数据高效多了。下面以 SQL Server2008 为后台数据库讲解如何实现分页显示数据。

实现数据的分页显示,需要关注以下几点。

1. 根据实际的页面设计，确定在数据列表中每次显示多少条数据，也就是说每次从数据库中需要查询出多少条数据用于页面显示。

2. 计算显示的页数，就是按照每页显示的数据量总共需要划分成多少页。

由于在页面中显示的数据数量是固定的，而数据库中总共存储了多少条数据是未知的，因此要想得到总页数，需要以下几个步骤。

(1) 首先要通过查询获取数据库中总的记录数，通过 SQL 提供了 count 聚合函数，就可以获取数据库中记录的总数，代码如示例 9-1 所示。

示例9-1

```
/**
 * 获取数据库中记录总数的代码
 */
    public int getCount(){
        String sql = "select count(*) from Notice";
        int count = 0;
        //省略执行代码
        if(rs.next()){
            count = rs.getInt(1);
        }
        return count;
    }
```

从示例 9-1 中的代码可以看到，当执行了使用 count 函数的 SQL 语句后，将获得 Notice 表中的数据总数，然后将其数据返回。

(2) 有了数据库记录总数后，我们就可以根据每页显示的记录数来计算总共需要划分为多少页，代码如示例 2 所示。

示例9-2

```
/**
 * 计算总页码
 */
    public int getTotalPages(int count, int pageSize){
        int totalpages = 0;
        totalpages =
(count % pageSize == 0)?(count/pageSize):(count/pageSize + 1);
        return totalpages;
    }
```

在示例 9-2 的代码中，使用了条件二元运算符“?”的方式进行数据处理。那么该条语句的执行结果是：如果“?”运算符前的表达式条件为真，则将运算符“:”前的表达式结果赋予变量 totalpage，否则就将运算符“:”后的表达式结果赋予变量 totalpage。

3. 编写 SQL 语句。

实现数据分页显示的关键就是如何编写 SQL 查询语句。下面我们将使用一种比较常

用的方式来编写 SQL 语句,代码如示例 9-3 所示。

示例9-3

```
/**
 * 分页 SQL 语句
 */
    String sql = "SELECT TOP 2 *
                 FROM Notice
                 WHERE (Nno NOT IN
                 (SELECT TOP 4 Nno
                 FROM Notice
                 ORDER BY NcreateTime))
                 ORDER BY NcreateTime";
```

示例 9-3 中的 SQL 语句,从结构上可以发现,使用了两层嵌套的查询方式,为什么使用两层的嵌套语句呢?每一层的语句起到了什么作用?下面我们就来具体分析每一语句的作用。

(1)内层的 select 语句是一条普通的查询语句,它执行结果是将 Notice 表按照时间降序排列选出前 4 条记录。

(2)第二层的 select 语句的执行结果是将 Notice 表按照时间降序排列选出前两条记录,not in 是这两条记录应不包含内层 select 语句选中的 4 条记录。这样也就得到了从第 5 条记录开始的 2 条记录。

那么我么可以总结出一个规律:

假定每页显示的记录数为 pageSize、当前页页码为 pageIndex,则在内层 select 语句中可以先查询出当前页码前的所有条记录,即 pageSize *(pageIndex－1)条记录,在外层 select 语句中在查询前 pageSize *(pageIndex－1)记录之后的 pageSize 条记录。如示例 9-4 所示。

示例9-4

```
/**
 * 分页 SQL 语句
 */
    String sql = "SELECT TOP pageSize *
                 FROM table
                 WHERE id NOT IN
                 (
                 SELECT TOP pageSize * (pageIndex - 1) id FROM table ORDER BY id
                 )";
```

在第 8 章中,我们已经学习了 MVC 的开发模式,用 Jsp＋Servlet＋Javabean 来实现,那么分页的实现我们就采用这种方式。

4. 分页在程序中的实现。

首先创建名为 Ch09_1 的 web 项目,导入包。

(1) 在 com. bean 中创建 Notice. java 实体类；

(2) 在 com. dao 中创建 NoticeDao. java ，ConnectionManager. java(工具类)；

(3) 在 com. biz 中创建 NoticeBiz. java；

(4) 在 com. servlet 中创建 NoticeServlet. java；

(5) 配置 web. xml 中的 servlet；

(6) 创建 ShowNotice. jsp 页面；

(7) 部署运行。

示例9-5

```
/**
 * Notice 实体类
 */
package com.bean;
import java.util.Date;
public class Notice {
    private int id;              //ID
    private String title;        //通知公告标题
    private String content;      //通知公告内容
    private String editor;       //通知公告发布者
    private Date createTime; //发布时间
    private int type;            //通知公告类型
    public Notice(){}
//..省略个属性的 set get 方法
}
```

NoticeDao. java 代码如下。

```
package com.dao;
import… …
public class NoticeDao {
  public int getCount(){
      Connection dbConnection = null;
      PreparedStatement pStatement = null;
      ResultSet res = null;
      public int getCount(){
      int count = 0;
      try {dbConnection = ConnectionManager.getConnection();
          String sql = "select count(*) from Notice";
          pStatement = dbConnection.prepareStatement(sql);
          res = pStatement.executeQuery();
          if(res.next()){
              count = res.getInt(1);
          }
      } //省略 catch finally
      return count;
  }
```

```
    public List getPage(int pageSize ,int pageIndex){
        List list = new ArrayList();
        int count = 0;
        try {dbConnection = ConnectionManager.getConnection();
            String sql = "SELECT TOP " + pageSize + " * FROM Notice WHERE (Nno NOT IN (SELECT
TOP " + pageSize * (pageIndex - 1) + " Nno FROM Notice ORDER BY NcreateTime)) ORDER BY
NcreateTime";
            pStatement = dbConnection.prepareStatement(sql);
            res = pStatement.executeQuery();
            while (res.next()) {
                Notice notice = new Notice();
                notice.setId(res.getInt("Nno"));
                notice.setTitle(res.getString("Ntitle"));
                notice.setContent(res.getString("Ncontent"));
                notice.setEditor(res.getString("Neditor"));
                notice.setCreateTime(res.getDate("NcreateTime"));
                notice.setType(res.getInt("Ntype"));
                list.add(notice);
            }
        } //省略 catch finally
        }
        return list;
    }
}
```

连接数据库工具类 ConnectionManager.java 前面已经介绍过，这里不再赘述，NoticeBiz.java 如下。

```
/**
 * 业务逻辑类 NoticeBiz
 */
package com.biz;
import java.util.List;
import com.dao.NoticeDao;
public class NoticeBiz {
    public int getTotalPages(int pageSize){
    NoticeDao nd = new NoticeDao();
    int count = nd.getCount();      //总记录数
    int totalpages = 0;
    totalpages = (count % pageSize == 0)?(count/pageSize):(count/pageSize + 1);
    return totalpages;
    }
    public List PageList(int pageSize,int pageIndex){
        NoticeDao nd = new NoticeDao();
        //总页数
        int totalPage = this.getTotalPages(pageSize);
        List list = nd.getPage(pageSize, pageIndex);
```

```
            return list;
        }
    }
```

```
    /**
     * 控制类 noticeServlet.java
     */
    package com.servlet;
    import……
    public class NoticeServlet extends HttpServlet {
        private int pageSize = 2;                       //每页显示两条记录
        private int pageIndex = 1;                      //当前页
        NoticeBiz nb = new NoticeBiz();
        public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
            int totalPages = nb.getTotalPages(pageSize); //获得总页数
            //确定当前页
            String currentPage = request.getParameter("pageIndex");
            if(currentPage == null){
                currentPage = "1";
            }
            pageIndex = Integer.parseInt(currentPage);
                if(pageIndex < 1){
                pageIndex = 1;
            }else if(pageIndex > totalPages){
                pageIndex = totalPages;
            }
            List list = nb.PageList(pageSize, pageIndex);
            request.setAttribute("pageIndex", pageIndex);
            request.setAttribute("totalPages", totalPages);
            request.setAttribute("list", list);request.getRequestDispatcher("showNotice.
    jsp").forward(request,response);
        }
        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            this.doGet(request, response);
        }
    }
```

```
    /**
     * web.xml 的配置
     */
        <servlet>
        <servlet-name>noticeServlet</servlet-name>
        <servlet-class>com.servlet.NoticeServlet</servlet-class>
      </servlet>
```

```
<servlet-mapping>
  <servlet-name>noticeServlet</servlet-name>
  <url-pattern>/noticeServlet</url-pattern>
</servlet-mapping>
```

```
/**
 * ShowNotice.jsp
 */
<%@ page language="java" import="java.util.*,com.bean.Notice" contentType="text/
html; charset=utf-8" pageEncoding="utf-8"%>
<html>
<head>
<title>show notice</title>
</head>
<body>
<a href="noticeServlet?pageIndex=1">首页</a>
<a href="noticeServlet?pageIndex=<%=(Integer)(request.getAttribute("pageIndex"))-1
%>">上一页</a>
<a href="noticeServlet?pageIndex=<%= (Integer)(request.getAttribute("pageIndex"))+1
%>">下一页</a>
<a href="noticeServlet?pageIndex=<%= request.getAttribute("totalPages")%>">末页</a>
(<%=request.getAttribute("pageIndex")%>/<%= request.getAttribute("totalPages")%>)
<%
  List list = (List)request.getAttribute("list");
  Iterator it = list.iterator();
  while(it.hasNext()){
    Notice notice = (Notice)it.next();
    out.print("<br/>");
    out.print(notice.getTitle());
}
%>
</body>
</html>
```

最后部署运行，打开 IE 浏览器，在地址栏中写入 http://localhost:8080/Ch09_1/noticeServlet 运行的效果如图 9-1 所示。



图 9-1 分页显示运行效果图

9.2 SmartUpload 实现文件上传

现在网络共享逐渐成为传递信息、共享资源的一种常用方式。用户将自己计算机中的文件上传至服务器端以便其他人浏览、欣赏，已经是 Web 项目中最常用的功能了。最典型的文件上传应用就是在日常生活中，很多人都有自己个性化的博客或者私人空间，可以将自己在日常工作、生活中的照片放到空间里，供亲朋好友浏览。

那么文件上传功能从技术上如何实现呢？实现文件上传，涉及对文件的读写操作，实现起来需要编写大量的代码，并容易引发异常。幸运的是，目前有很多非常实用的文件上传工具，可以帮助我们实现文件上传的功能，其中应用比较多的就是 SmartUpload 组件，使用该组件可以极大地简化开发人员的编码工作量，接下来我们就重点学习如何使用 SmartUpload 组件在 JSP 中实现文件上传功能。

9.2.1 SmartUpload 简介

SmartUpload 组件是一个实现文件上传的免费组件，虽然已不再提供更新，但由于使用简单方便依然被非常广泛地使用。使用 SmartUpload 组件具有以下几个特点：

- 使用简单 SmartUpload 组件可以方便地嵌入到 JSP 文件中，在 JSP 文件中仅编写少量代码即可完成文件的上传和下载功能，十分方便；
- 能够全程控制上传内容：使用 SmartUpload 组件提供的对象及操作方法，可以获得全部上传文件的信息，包括文件名称、类型、大小等，方便操作；
- 能够对上传文件的大小、类型进行控制：为了避免在上传过程中出现异常数据，在 SmartUpload 组件中，专门提供了相应的方法用于限制不符合要求的文件数据。

9.2.2 表单的属性设置

以前我们获取表单数据的方式是采用 request 对象的 getParameter ()方法，这种方法只能获取表单提交的文本信息，却无法获取文件数据。这是因为一般的文本类型等传至服务器的编码方式与文件传至服务器的编码方式是不同的，文件传至服务器必须使用 multipart/form-data 编码方式，因为使用的编码方式不同，所以使用 getParameter ()方法不能获取数据，所以我们需要在表单属性中添加属性 enctype，该属性的设置方法如下：

```
<form enctype="multipart/form-data" method="post">
```

注意上传文件时 form 标签的 method 属性必须取值为"post"，不能为" get"。

9.2.3 使用 File 控件选择文件

在学习 HTML 时，我们曾经学习过通过设置<input>标签的 type 属性可以实现在页面添加不同类型的控件，例如，文本输入框、按钮等。现在如果要实现文件上传，我们就要在页面中使用 File 控件。在表单中添加 File 控件的代码如示例 9-6 所示。

示例9－6

```
/**
 * 文件上传 form
 */
<form enctype="multipart/form-data" method="post">
    选择文件:<intput type="file" name="nfile">
</form>
```

运行示例9-6的代码,效果如图9-2所示。

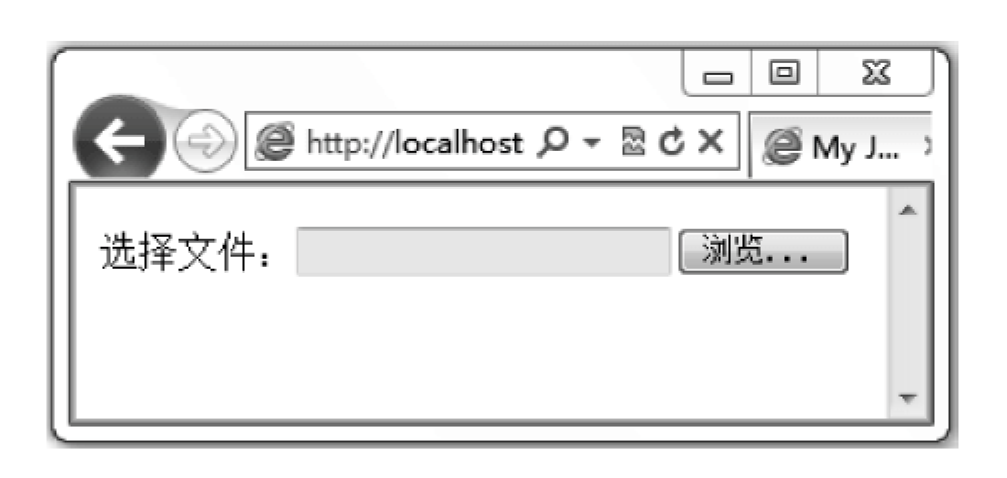


图9-2 File控件

使用File控件可以实现在本地进行文件选择,当用户单击"浏览"按钮后,会打开如图9-3所示的界面,让用户选择需要加载的文件。



图9-3 选择加载的文件

当用户选择了一个文件,并单击"打开"按钮后,用户在本地所选择的文件路径将在File控件的文本框中显示,如图9-4所示。

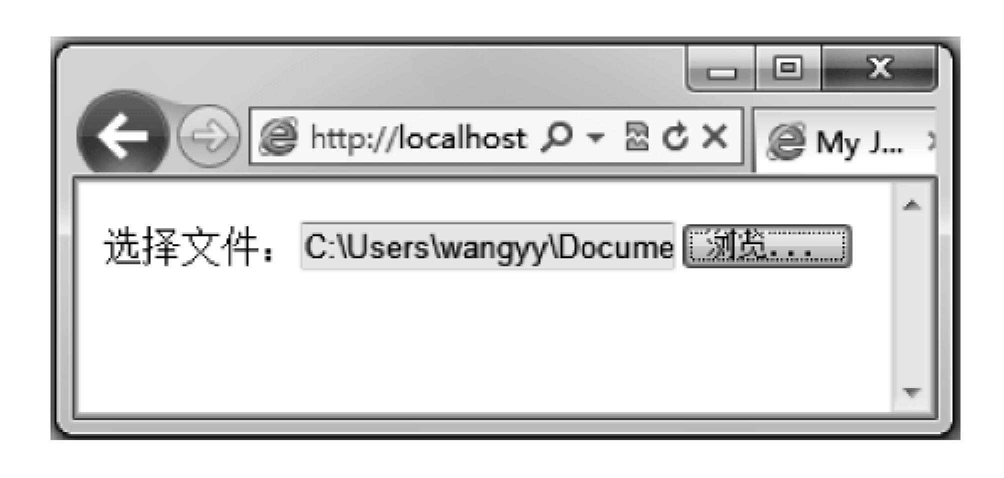


图9-4 加载文件

9.2.4 SmartUpload 组件的常用方法

如果希望在项目中使用 SmartUpload 组件，首先需要在项目中添加 smartupload.jar 文件。添加完 jar 文件后，在 JSP 文件中还需要将 SmartUpload 组件所使用的类库导入到 JSP 文件中。语法格式如下：

```
<%@ page import="com.jspsmart.upload.*" %>
```

了解了如何在项目中添加 SmartUpload 组件，在使用其完成文件上传之前，我们再来了解一下，该组件都提供了哪些对象及方法来实现文件的上传和下载。

1. File 类

File 类的作用是封装上传文件所包含的所有信息。通过调用 File 类的方法，我们可以方便地获取到有关上传文件的信息，例如，文件的名称、文件的大小、文件的类型及文件内容数据等。

注意

File 类所包含的文件信息是单个文件，而不是所有上传文件的信息。

File 类提供的常用方法如表 9-1 所示。

表 9-1 File 类的常用方法

方法名称	功能描述
public void saveAs(String destFilePathName)	将文件保存，参数 destFilePathName 是保存的文件名
public void saveAs(String destFilePathName, int optionSaveAs)	将文件保存，参数 destFilePathName 是保存的文件名，参数 optionSaveAs 表示保存的选项
public boolean isMissing( )	判断用户是否选择了文件，即对应的表单项是否为空，返回值为 boolean 类型
public String getFiledName( )	获取表单中当前上传文件所对应的表单项的名称
public String getFileName( )	获取上传文件的文件名称，不包含路径
public String getFilePathName( )	获取文件的全名称，包含路径的完整文件名称
public String getFileExt()	获取文件的扩展名
public String getContentString( )	获取文件的内容，返回值为字符串类型
public int getSize( )	获取文件的大小，单位字节，返回值为 int 类型

2. Files 类

Files 类与 File 类的区别在于 File 类包含了单个上传文件的信息，而 Files 类表上传文件的集合，通过它可以得到上传文件的数量、大小等信息。Files 类提供的常用方法如表 9-2 所示。

表 9-2 Files 类的常用方法

方法名称	功能描述
public int getCount()	取得文件上传的数目
public File getFile(int index)	取得指定位置的 File 文件对象
public long getSize()	取得上传文件的总长度
public Collection getCollection()	将所有上传文件对象以 Collection 的形式返回

3. SmartUpload 类

SmartUpload 类用于实现文件的上传与下载操作，SmartUpload 类提供的常用方法如表 9-3 所示。

表 9-3 SmartUpload 类的常用方法

方法名称	功能描述
public final void initialize（PageContext pageContext）	执行上传和下载的初始化工作，必须实现
public void upload()	实现文件数据的上传，放在 initialize 方法后
public int save(String pathName)	将全部上传文件保存到指定的目录下，并返回保存的文件个数
public void setAllowFilesList（String ExtList）	指定允许上传的文件扩展名，接收一个扩展名列表，以逗号分隔
public void setDeniedFilesList（String fileList）	指定禁止上传的文件扩展名列表，每个扩展名之间以逗号分隔
public void setMaxFileSize(long filesize)	设定每个文件允许上传的最大长度
public void setTotalMaxFileSize（long totalfilesize）	设定允许上传文件的总长度

9.2.5 SmartUpload 组件的应用

了解了 SmartUpload 组件的相关对象及方法，下面我们就进一步学习如何在 JSP 中用 SmartUpload 组件实现文件上传的功能。

示例9-7

```
/**
 * upload.jsp 下载页面
 */
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ page import="com.jspsmart.upload.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>上传页面</title>
  </head>
  <body>
    <form action="do_upload.jsp" enctype="multipart/form-data" method="post">
        选择文件 1:<input type="file" name="file1">
        <br/>
        选择文件 2:<input type="file" name="file2">
        <br>
        <input type="submit" value="上传">
    </form>
  </body>
</html>
```

此处 form 提交到一个 JSP 页面处理，当然也可以提交到一个 servlet 来处理。

```
/**
 * do_upload.jsp 处理下载页面
 */
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ page import="java.util.*,com.jspsmart.upload.*" errorPage=""%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>文件处理上传页面</title>
  </head>
  <body>
<%
    SmartUpload su = new SmartUpload();
    su.initialize(pageContext);
    su.setCharset("UTF-8");
    su.setMaxFileSize(100000);
    su.setTotalMaxFileSize(200000);
    su.setAllowedFilesList("doc,txt");
    su.setDeniedFilesList("exe,bat,jsp,html");
    su.upload();                              //上传文件
    int count = su.save("/upload");
    out.println(count+"文件上传成功");
    for(int i=0;i<su.getFiles().getCount();i++){
        File file = su.getFiles().getFile(i);
        if(file.isMissing())continue;         //若文件不存在则继续
        String filepath =request.getRealPath("/")+"upload"+"\\"+file.getFileName();
        //显示当前文件信息
        out.println("<table border=1>");
        out.println("<tr><td>表单项名(FiledName)</td><td>"+file.getFieldName()+"</
td></tr>");
        out.println("<tr><td>文件长度(FileSize)</td><td>"+file.getSize()+"</td></
tr>");
        out.println("<tr><td>文件名(FileName)</td><td>"+file.getFileName()+"</td>
</tr>");
        out.println("<tr><td>文件扩展名(FileExt)</td><td>"+file.getFileExt()+"</td
></tr>");
        out.println("<tr><td>文件全名(FilePathName)</td><td>"+filepath+"</td></tr
>");
        out.println("</table></br>");
    }
 %>
</body>
</html>
```

示例 9-17 的运行过程及效果分别如图 9-5 及图 9-6 所示。

上传成功后在 tomcat 的\webapps\Ch09_2\upload 目录中即出现上传文件。

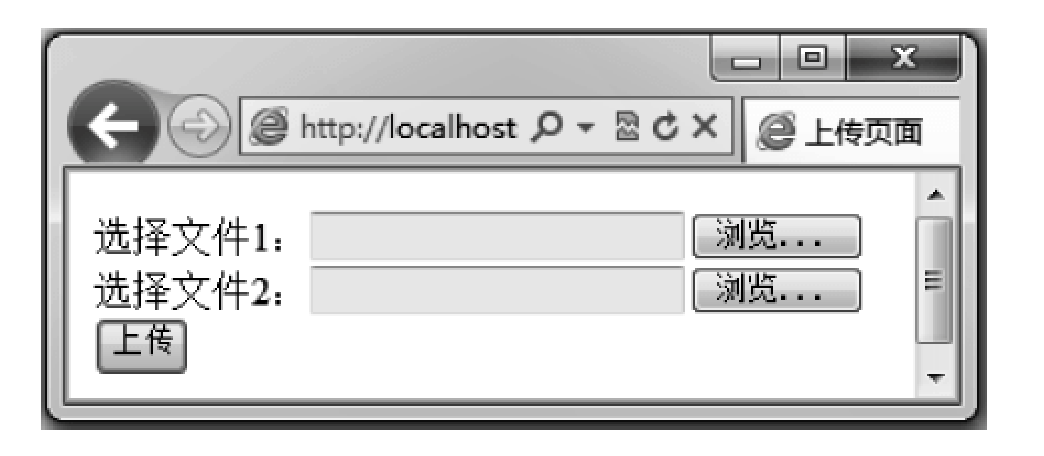
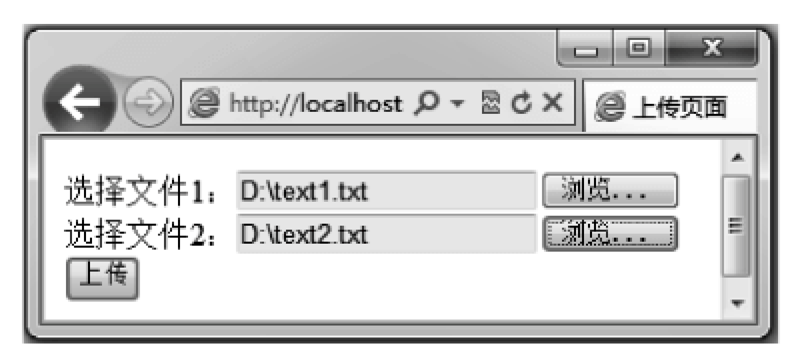
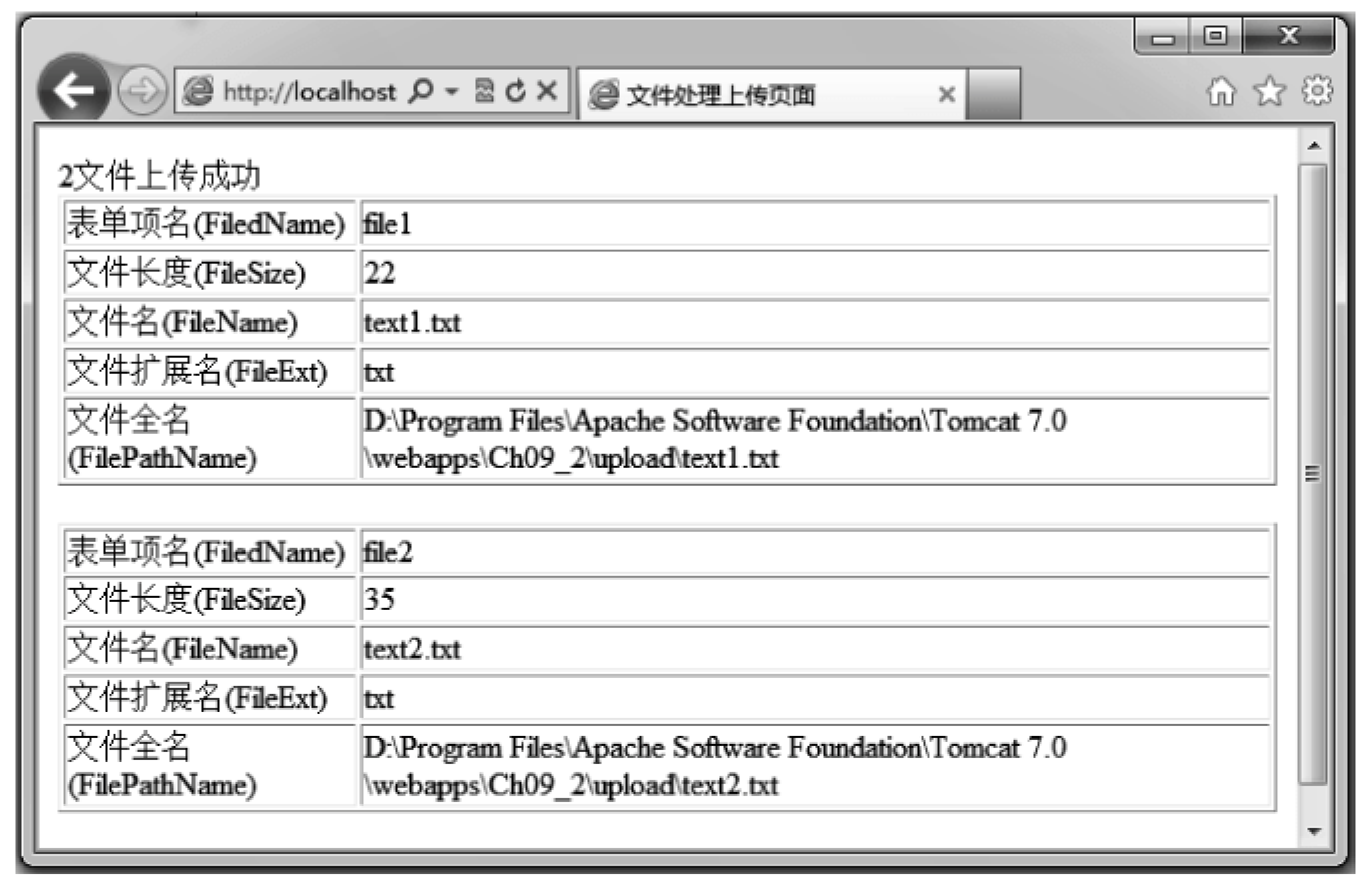


图 9-5 文件上传





表单项名(FiledName)	file1
文件长度(FileSize)	22
文件名(FileName)	text1.txt
文件扩展名(FileExt)	txt
文件全名(FilePathName)	D:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\Ch09_2\upload\text1.txt

表单项名(FiledName)	file2
文件长度(FileSize)	35
文件名(FileName)	text2.txt
文件扩展名(FileExt)	txt
文件全名(FilePathName)	D:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\Ch09_2\upload\text2.txt

图 9-6 处理文件上传后页面

9.3 SmartUpload 实现文件下载

学习了 SmartUpload 组件实现文件上传的功能,接下来我们学习文件下载。

示例9-8

```
/**
 * download.jsp 下载页面
 */
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ page import="com.jspsmart.upload.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>下载文件</title>
  </head>
```

```
  <body>
    单击相应的链接下载<br>
    <a href="do_download.jsp">text1.txt</a>
  </body>
</html>
```

```
/**
 * do_download.jsp 处理下载页面
 */
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@ page import="com.jspsmart.upload.*"%>
<%
    out.clearBuffer();
    out = pageContext.pushBody();
    //新建一个 SmartUpload 对象
    SmartUpload su = new SmartUpload();
    //初始化
    su.initialize(pageContext);
    //设定 ContentDisposition 为空时禁止浏览器自动打开文件
    su.setContentDisposition(null);
    //下载文件
    su.downloadFile("/upload/text1.txt");
%>
```

经验

在代码中若没有

```
out.clearBuffer();
out = pageContext.pushBody();
```

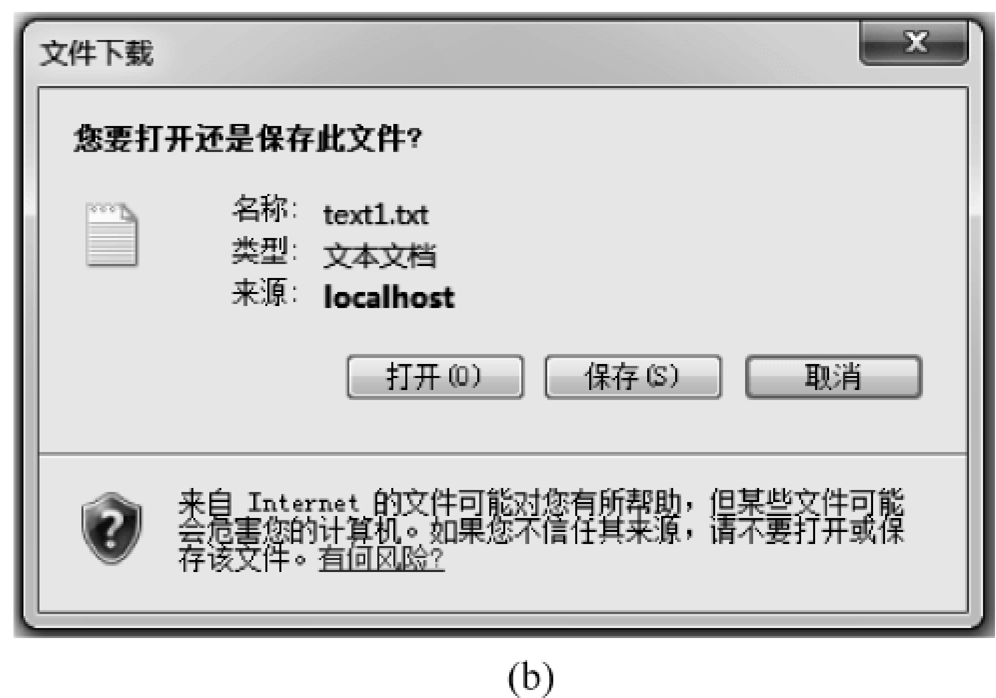
则下载时出现异常

```
java.lang.IllegalStateException: getOutputStream() has already been called for this response
```

示例 9-8 的运行效果如图 9-7(a)、(b)所示。



(a)



(b)

图 9-7 文件下载

9.4 上 机 练 习

1. 实现通知公告发布系统列表的分页显示。

需求说明：在通知公告发布系统的后台，在页面右部动态显示通知公告列表时加上分页显示功能。

实现思路：参照 9.1 节。

实现如图 9-8 所示。



图 9-8 分页效果图

2. 实现文件上传、下载。

需求说明：新建项目 CH09_2，实现文件的上传与下载功能。

实现思路：参照 9.2、9.3 节。

9.5 总 结

（1）实现数据分页显示，需要经过的步骤如下：

- 确定每页显示的数据数量；
- 确定分页显示所需的总页数；
- 编写 SQL 查询语句，实现数据查询；
- 在 JSP 页面中进行分页显示设置。

（2）SmartUpload 组件是实现文件上传功能的免费组件，可以在 JSP 中实现文件的上传与下载功能。

(3) 在文件上传表单页面中,需要设置表单属性 enctype= "multipart/form-data"提交方式 method= " post"。

9.6 作　　业

一、选择题

1. 声明 SmartUpload 对象的正确方法是(　　)。

A. Smart Upload su=new SmartUpload ()

B . SmartUpload su= SmartUpload. new Instance ()

C. SmartUpload su= SmartUpload. initialize ()

D. SmartUp load 无须实例化,可直接使用

2. 下面不属于分页实现步骤的是(　　)。

A. 确定每页显示的数据数量

B. 计算总页数

C. 编写查询 SQL 语句

D. 使用下拉列表显示页数

3. 使用 SmartUpload 实现文件上传时,对以下方法描述错误的是(　　)。

A. 使用 setAllowedFilesList 方法可以指定允许上传的文件类型列表

B. 使用 save 方法可以将上传文件保存到指定目录下

C. 使用 setTotalMaxFileSize 方法可以指定每个文件上传的最大长度

D. 使用 setDeniedFilesList 方法可以指定禁止上传的文件类型列表

二、简答题

1. 简述实现分页的步骤?

2. 写出 SQL Servler2008 实现分页查询的 SQL 语句?

3. 在文件上传时,如何控制上传文件的大小和类型?

三、程序题

编写一个 web 项目,显示你的个人信息,要求能够上传你的照片,格式为 jpg 或 gif,图片大小不超过 3M。

第 10 章　JSP 高级程序设计

本章学习目标

- 理解并会使用 EL 表达式
- 理解并会使用常用的 JSTL 标签
- 了解框架技术

本章主要介绍 JSP 高级程序设计的相关技术，主要包括 EL 表达式、JSTL 标准标签库以及 Java Web 开发中应用的框架技术。

10.1　EL 表达式

在前面的学习中可以看到，JSP 页面仍然嵌入很多 Java 代码，不利于对表现层的维护和更新。接下来我们介绍如何使用 JSTL 标签库和 EL 表达式，实现无 Java 代码嵌入的 JSP 页面开发。

10.1.1　什么是 EL 表达式

EL 的全称是 Expression Language，它是一种借鉴了 JavaScript 和 XPath 的表达式语言。EL 定义了一系列的隐含对象和操作符，使开发人员能够很方便地访问页面的上下文，以及不同作用域内的对象，而无需在 JSP 页面嵌入 Java 代码。EL 表达式通常用于在某个作用域(page、request、session、application 等)内取得属性值，或者做简单的运算和判断。EL 表达式有以下特点。

1. 自动转换类型

使用 EL 得到某个数据时可以自动转换类型，因此对于类型的限制更加宽松。

2. 使用简单

与 JSP 页面中嵌入的 Java 代码相比，EL 表达式使用起来更加简单。

10.1.2　EL 简介

1. 语法结构

```
${EL 表达式}
```

其中“$”与“{}”缺一不可。

2. “[ ]”与“.”操作符

EL 提供“.”和“[ ]”两种操作符来存取数据。

EL 表达式通常由两部分组成：对象和属性。就像在 Java 代码中一样，在 EL 表达式中也可以用点操作符“.”访问对象的某个属性，例如，通过 ${user.name}可以访问 user 对象的 name 属性；而通过 ${ goods.supplier.address}则可以访问商品供货商的地址。

与点操作符类似，“[ ]”操作符也可以访问对象的某个属性，如 ${user['name']}，除此之外。“[ ]”操作符还提供了更加强大的功能。

- 当属性名中包含了特殊字符如“.”或“-”等的情况下，就不能使用点操作符来访问，这时只能使用“[ ]”操作符。
- 访问数组，如果有一个对象名为 array 的数组，那么我们可以根据索引值来访问其中的元素，如 ${array[0]}、${ array[1]}等

3. EL 作用域访问对象

通过学习 JSP 相关知识，我们知道 JSP 提供了 page、request、session、application 等 9 个内置对象。这些内置对象无需声明，就可以很方便地在 JSP 页面脚本中使用。相应地，在 EL 表达式语言中也提供了一系列可以直接使用的隐式对象。这里我们主要来讲解作用域访问对象。如图 10-1 所示。

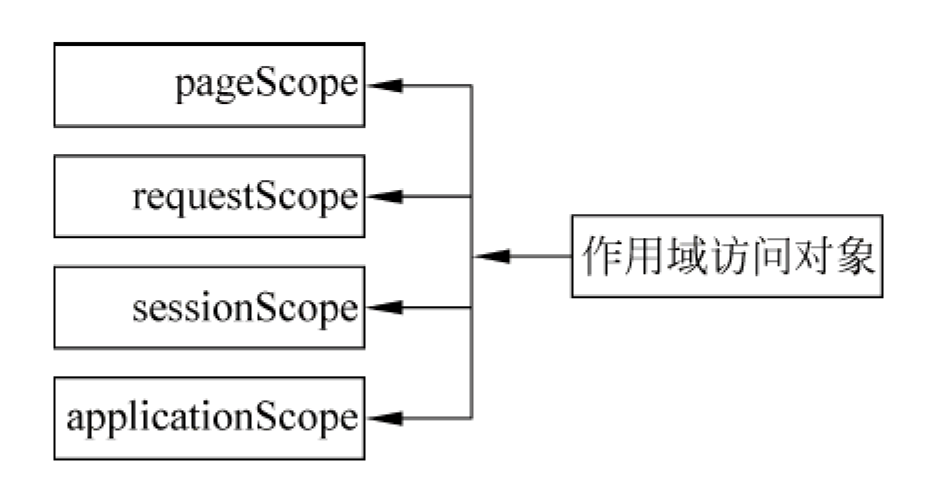


图 10-1　EL 作用域访问对象

EL 存取变量数据的方法很简单，例如：${username}。它的意思是取出某一范围中名称为 username 的变量。因为我们并没有指定哪一个范围的 username，所以它会依序从 page、request、session、application 范围查找。假如途中找到 username，就直接回传，不再继续找下去，但是假如全部的范围都没有找到时，就回传 null。例如：

```
request.setAttribute("user",user);
```

则可用 ${requestScope.user}访问 request 范围中的 user 对象，用 ${requestScope.user.name}访问 request 范围中的 user 对象的 name 属性。

10.2　JSTL 标签

10.2.1　JSTL 标签简介

通过使用 EL 表达式，在 JSP 页面中可以很容易地输出信息，从而在一定程度上简化了 JSP 页面开发的复杂度。但是由于 EL 表达式不能实现复杂逻辑的处理，在 JSP 页面中依然存在用 Java 代码处理表示层逻辑的现象。那么有没有一种技术，使我们既不用嵌入 Java 代码，又能在 JSP 中控制程序流程呢？那就是 JSTL 标签。

JSTL 的全称是 Java Server Pages Standard Tag Library，即 JSP 标准标签库。它包含了我们在开发 JSP 页面时经常用到的一组标准标签。JSTL 标签库包含了各种标签，如通用标签、条件判断标签和迭代标签等。

在开发中使用 JSTL 标签库，我们需要执行如下两个步骤。

1. 在项目中引用 JSTL 的两个 Jar 包和标签库描述符文件(.tld 文件)。

需要导入的两个 Jar 包为 jstl.jar 和 standard.jar。除此以外，标签库描述符文件也是必需的，这些资源都能在网上下载得到。

在集成开发环境 MyEclipse 中已经集成了 JSTL，因此这一过程可以由工具代为实现，方法如下。

首先在新建 web 项目时，弹出 New Web Project 对话框。在该对话框中的 J2EE Specification Level 选项组中选择 Java EE 5 0 单选按钮，MyEclipse 会自动在项目中添加 JSTL 所需要的 Jar 包和标签库描述符文件。而如果选择更低的版本，则需要选择 Add JSTL libraries to WEB-INF /lib folder 复选框，然后单击“Finish”按钮。

2. 在需要使用 JSTL 的 JSP 页面上使用 taglib 指令导入标签库描述符文件，导入核心标签库

例如：

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

10.2.2 JSTL 核心标签库

核心标签库在 JSTL 中占有十分重要的地位，按功能的不同可以分为通用标签库、条件标签库、迭代标签库等。下面我们分别来介绍

1. 通用标签库

通用标签用于在 JSP 页面内设置、删除和显示变量，它包含三个标签 <c:set>、<c:value>、<c:out>，接下来我们一一介绍。

(1) <c:set>标签用于定义变量，并将变量存储在 JSP 范围中或者 JavaBean 属性中，其语法格式分为如下两种。

① 语法

```
<c:set var="variable" value="v" scope="scope"/>
```

- var 属性的值是设置的变量名；
- value 属性的值是赋予变量的值；
- scope 属性对应的是变量的作用域，可选值有 page、request、session 和 application。

例如，在请求范围内将变量 currentIndex 设置为 8，用<c:set>标签可以写为

```
<c:set var="currentIndex" value="8" scope="request"/>
```

② 将 value 值存储到 target 对象的属性中

语法

```
<c:set value="value" target="target" property="property" />
```

- target 属性是操作的对象，可以使用 EL 表达式表示；
- property 属性对应对象的属性名；
- value 属性是赋予对象属性的值。

（2）<c：out>标签用来显示数据的内容，类似于 JSP 中的<%＝%>。但是功能更加强大，代码也更加简洁，方便页面维护。其语法格式分为指定默认值和不指定默认值两种形式。

① 不指定默认值　语法如下：

```
<c:out value="value" />
```

value 属性指需要输出的值，可以用 EL 表达式输出某个变量。

② 指定默认值　语法如下：

```
<c:out value="value" default="default" />
```

default 属性是 value 属性的值为空时，输出的默认值。

下面用一个示例来加深对<c:set>和<c：out>标签的理解，代码如示例 10-1 所示。

示例 10－1

```
/**
 * JSTL 标签
 */
  <%
      Notice notice = new Notice();
    request.setAttribute("notice",notice);
 %>
<c:set target="${notice}" property="title" value="defaultTitle"></c:set>
<c:out value="${notice.title}" default="noTitle"></c:out>
```

在该示例中 Notice 类是一个 JavaBean。我们首先使用<c：set>标签为 notice 对象的 title 属性设置一个值，然后使用<c:out>标签输出该属性的值，如果该属性值为空，则显示“noTitle”。

（3）<c：remove>标签与<c:set>标签的作用相反，<c：remove>用于移除指定范围的变量，语法格式如下：

```
<c:remove var="value" scope="scope"/>
```

- var 属性是指待删除的变量的名称；
- scope 属性是指删除的变量所在的范围，可选项有 page、request、session、application。如果没有指定，则默认为 page。

示例 10－2

```
/**
 * JSTL 的<c:remove>应用
 */
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" %>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>使用 JSTL 设置变量</title>
  </head>
  <body>
      <!-- 设置之前应该是空值 -->
        设置变量之前的值是: message=<c:out value="${message}" default="null"/><br>
        <!-- 给变量 message 设值 -->
        <c:set var="message" value="Hello JSP!" scope="page"></c:set><!-- 此时
message 的值应该是上面设置的"已经不是空值了" -->
        设置新值以后: message=<c:out value="${message}"></c:out><br>
        <!-- 把 message 变量从 page 范围内移除 -->
        <c:remove var="message" scope="page"/>
        <!-- 此时 message 的值应该显示 null -->
        移除变量 message 以后: message=<c:out value="${message}" default="null"></c:
out>
  </body>
</html>
```

该示例的显示效果如图 10-2 所示。



图 10-2 用 JSTL 设置变量

2. 条件标签

对于包含动态内容的 Web 页面,不同类型的用户看到不同形式的内容。例如,对于一个资料下载网页,经常会根据用户的积分多少,判断是否可以看到下载链接,这时就要用到 JSTL 的另外一个常用的标签 <c:if>条件标签。<c: if>标签用来执行流程的控制,其功能和 Java 语言中的 if 完全相同,其语法结构为

```
<c:if test="condition" var="varName" scope="scope">
    内容…
</c:if>
```

test 属性是此条件标签的判断条件,当 test 中表达式的结果为 true 时,会执行内容,如果为 false 则不会执行。

- var 属性定义变量,该变量存放判断以后的结果,该属性可以省略。
- scope 属性是指 var 定义变量的存储范围,可选值有 page、request、session 和 application,该属性可以省略。

示例10-3

使用<c:if>标签完成登录后，如果用户已经登录则显示成功的消息，否则显示登录form。

```
/**
 * 用 JSTL EL 完成登录
 */
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
         <title>登录页面</title>
    </head>
    <body>
        <c:set var="loggedIn" value="${not empty sessionScope.userId}"/>
        <c:if test="${not loggedIn}">
        <form id="login" method="post" action="loginServlet">
                用户名:<input id="userName" name="userName" type="text"><br>
                密 码:<input id="passWord" name="passWord" type="password"><br>
                    <input type="submit" value="登录">
        </form>
        </c:if>
        <c:if test="${loggedIn}">
                您已经登录!
        </c:if>
    </body>
</html>
```

```
/**
 * 没有使用 JSTL EL 完成的登录
 */
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登录页面</title>
  </head>
    <body>
    <%
    String userId = (String)session.getAttribute("userId");
    if (userId == null) {
    %>
    <form id="login" method="post" action="loginServlet">
            用户名:<input id="userName" name="userName" type="text"><br>
            密 码:<input id="passWord" name="passWord" type="password"><br>
```

```
            <input type="submit" value="登录">
        </form>
    <%
    } else {
    %>
        您已经登录!
    <%
    }
    %>
  </body>
</html>
```

通过对比,显然<c:if>标签简化了工作量,而且使代码结果看起来更加清晰,易于管理和维护。

3. 迭代标签

在JSP的开发中,迭代是经常要使用到的操作。例如,列表显示查询结果等。在早期的JSP中,通常使用Java代码来实现集合对象(如List、Iterator等)的遍历。现在,通过JSTL的<c:forEach>标签,能在很大程度上简化迭代操作。

<c:forEach>标签有两种语法格式,一种用来遍历集合对象的成员,一种用来使语句循环执行指定的次数。

(1) 遍历集合对象的成员,其语法格式如下:

```
<c:forEach var="varName" items="lcollectionName" varStatus="varStatusName" begin=
"beginlndex" end="endlndex" step="step">
…内容
</c:forEach>
```

- var属性是对当前成员的引用,即如果当前循环到第一个成员,那么var就引用第一个成员,如果当前循环到第二个成员,它就引用第二个成员,以此类推;
- tems指被迭代的集合对象;
- varStatus属性用于存放var引用的成员的相关信息,如索引等;
- begin属性表示开始位置,默认为0,该属性可以省略;
- end属性表示结束位置,默认为集合的长度,该属性可以省略;
- step表示循环的步长,默认为1,该属性可以省略。

(2) 指定语句的执行次数,其语法格式为

```
<c:forEach var="varName" varStatus="vatStatusName" begin="beginlndex" end="endlndex"
step="step">
…内容
</c:forEach>
```

- var属性是对当前成员的引用;
- varStatus属性用于存放var引用的成员的相关信息,如索引等;
- begin属性表示开始位置,默认为0,该属性可以省略;
- end属性表示结束位置;

• step 属性表示循环的步长，默认为 1，该属性可以省略。

格式 2 与格式 1 的区别是：格式 2 不是对一个集合对象遍历，而是根据指定的 begin 属性、end 属性以及 step 属性执行本题内容固定的次数。

迭代标签在实际开发中的应用非常广泛，例如，示例 10-4 来显示通知的信息。

示例10－4

```
/**
 * 用 JSTL EL 完成分页显示 notice
 */
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="java.util.*" %>
<%@ page import="com.bean.Notice" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>show notice</title>
</head>
<body>
<table border="1">
    <tr>
        <th>通知标题</th>
    </tr>
<c:forEach var="notice" items="${requestScope.list}" varStatus="status">
    <tr <c:if test="${status.index%2==1}"> style="background-color:rgb(219,241,212);"</c:if>>
        <td>${notice.title}</td>
    </tr>
</c:forEach>
</body>
</html>
```

效果如图 10-3 所示。



图 10-3　用迭代显示通知标题

10.3 框架技术

在开发 JSP 程序时，采用合适的开发框架可以很好地提高开发效率。Struts、Spring 和 Hibernate 是 Java Web 开发中比较优秀的开源框架，这些框架各有特点，各自实现了不同的功能，在开发的过程中，如果能够将这些框架集成起来，将会使开发过程大大简化，很大程序上降低开发成本。

下面将对 Struts、Spring 和 Hibernate 进行简要介绍。有兴趣的读者请参照这些方面的资源来学习，从而提高自己的开发能力。

10.3.1 Struts 框架

Struts 框架是 Apache 组织的一个开放源代码项目，它是采用 Java Servlet 和 JSP 技术来构建基于 MVC 体系结构的 Web 应用程序的框架。Struts 框架具有良好的架构和设计、可重用、模块化、扩展性强等特点，因此已经被广泛应用于 Web 应用开发。

Struts 框架是目前非常流行的基于 Java 技术开发的 JSP Web 应用开发框架，它遵循了 MVC 设计模式。在 Struts 框架中，模型由实现业务逻辑的 JavaBean 组件构成，控制器由 ActionServlet 和 Action 来实现，视图由一组 JSP 文件与 Struts 标签库构成。

10.3.2 Spring 框架

Spring 是一个开源的框架，由 RodJohnson 创建，从 2003 年初正式启动。它能够降低开发企业应用程序的复杂性，可以使用 Spring 替代 EJB 开发企业级应用，而不用担心工作量太大、开发进度难以控制和复杂的测试过程等问题。它以 IOC（反向控制）和 AOP（面向切面编程）两种先进的技术为基础，完美地简化了企业级开发的复杂度。

Spring 框架主要由核心模块、上下文模块、AOP 模块、DAO 模块、Web 模块等 7 大模块组成，它们提供了企业级开发需要的所有功能，而且每个模块都可以单独使用，也可以和其他模块组合使用，灵活且方便的部署可以使开发的程序更加简洁灵活。Spring 的 7 个模块的部署如图 10-4 所示。

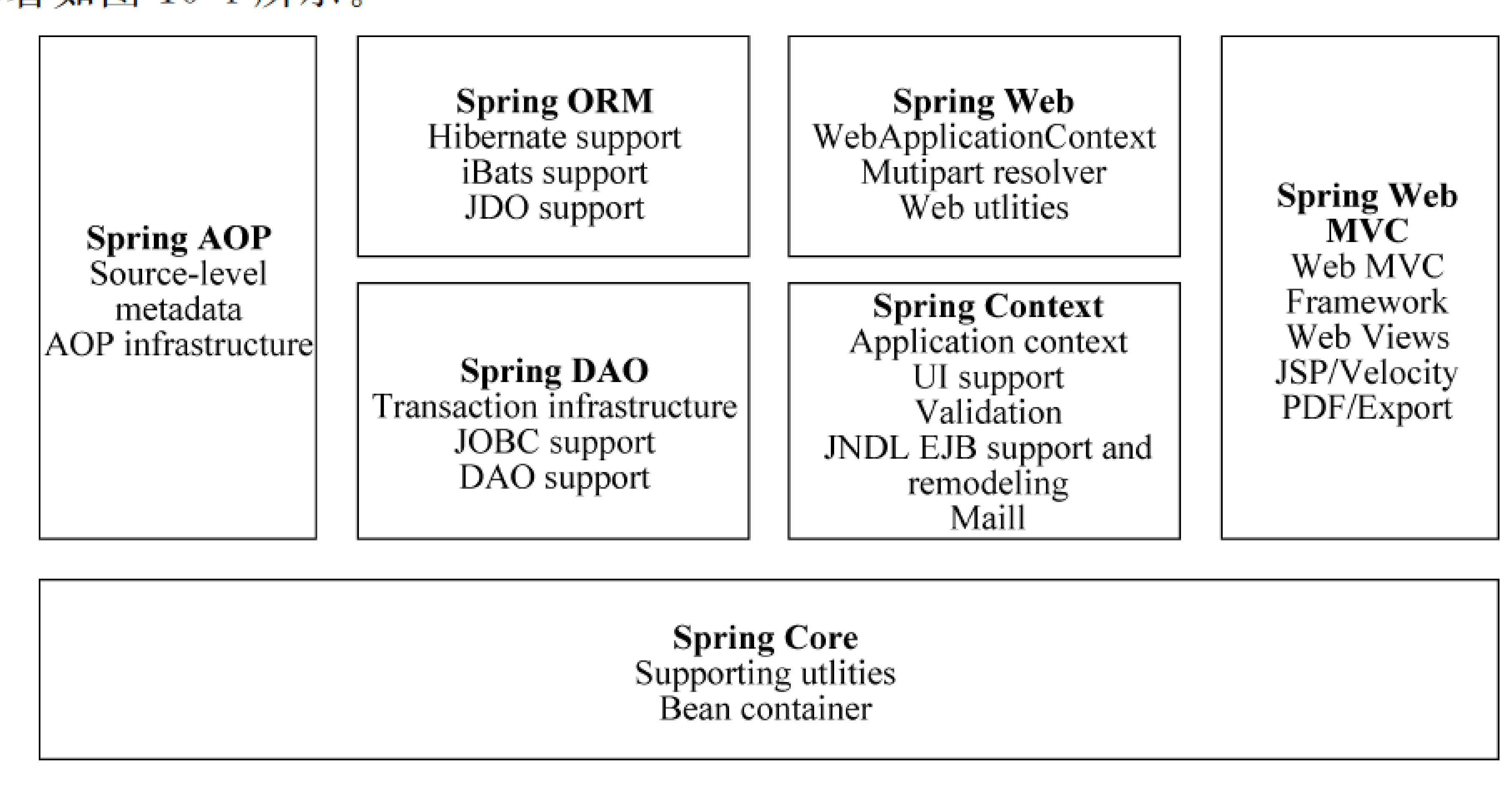


图 10-4 Spring 的 7 个模块

10.3.3 Hibernate 框架

Java 是一种面向对象的编程语言，但是通过 JDBC 方式操作数据库，运用的是面向过程的编程思想，为了解决这一问题，提出了对象—关系映射（Object Relational Mapping，ORM）模式。通过 ORM 模式，可以实现运用面向对象的编程思想操作关系型数据库。Hibernate 技术为 ORM 提供了具体的解决方案，实际上就是将 Java 中的对象与关系数据库中的表做一个映射，实现它们之间自动转换的解决方案。

Hibernate 在原有 3 层架构（MVC）的基础上，从业务逻辑层又分离出一个持久层，专门负责数据的持久化操作，使业务逻辑层可以真正地专注于业务逻辑的开发，不再需要编写复杂的 SQL 语句，增加了持久层的软件分层结构，具体关系如图 10-5 所示。

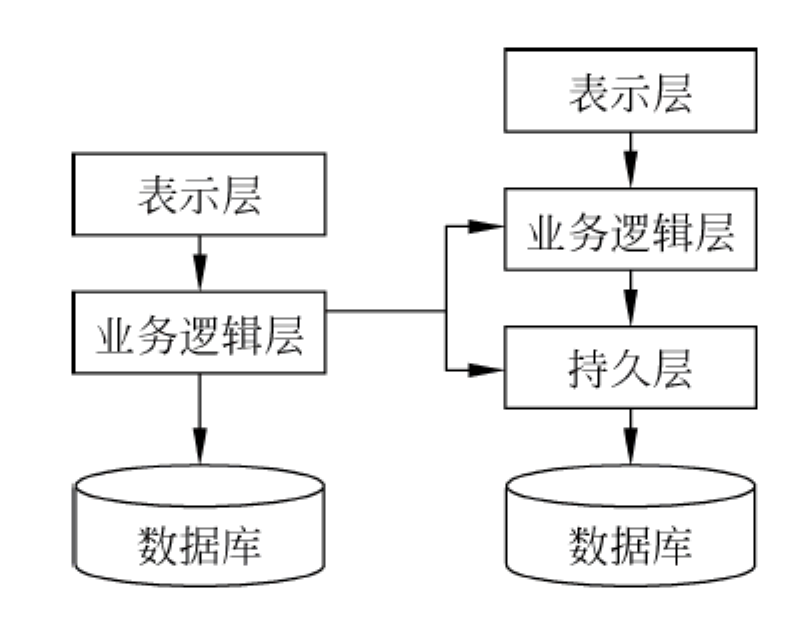


图 10-5 增加了持久层的软件分层结构

Hibernate 在 Java 对象与关系数据库之间起到了一个桥梁的作用，负责两者之间的映射，在 Hibernate 内部还封装了 JDBC 技术，向上一层提供面向对象的数据访问 API 接口。Hibernate 特点如下：

（1）它负责协调软件与数据库的交互，提供了管理持久性数据的完整方案，让开发者能够专著于业务逻辑的开发，不再需要考虑所使用的数据库及编写复杂的 SQL 语句，使开发变得更加简单和高效；

（2）应用者不需要遵循太多的规则和设计模式，让开发人员能够灵活的运用；

（3）Hibernate 支持各种主流的数据源，目前所支持的数据源包括 DB2，MySQL、Oracle、SQL Server 和纯 Java 驱动程序等；

（4）它是一个开放源代码的映射框架，对 JDBC 只做了轻量级的封装，让 Java 程序员可以随心所欲的运用面向对象的思想操纵数据库，无须考虑资源的问题。

10.4 上机练习

1. EL 表达式和 JSTL 标签库应用。

需求说明：利用 EL 表达式和 JSTL 标签库修改通知公告发布系统的前台 JSP 页面，使的页面变得简洁。

实现思路：参考 10.1 和 10.2 节。

2. EL 表达式和 JSTL 标签库应用。

需求说明：利用 EL 表达式和 JSTL 标签库修改通知公告发布系统的后台 JSP 页面，使

页面变得简洁。

实现思路：参考 10.1 和 10.2 节。

10.5 总　　结

(1) EL 表达式的语法有两个要素 $ 和{}，二者缺一不可。

(2) EL 表达式具有类型无关性，可以使用“.”或者“[ ]”操作符在相应的作用域(page、request、session、application)中取得某个属性的值。

(3) EL 表达式提供了 pageScope、requestScope、sessionScope、applicationScope 等隐式对象。

(4) JSTL 核心标签库中常用的标签有如下三类：

- 通用标签<c:set>、<c:out>、<c:remove>
- 条件标签<c: if>
- 迭代标签 <c : forEach>

(5) Java Web 开发中 Struts、Spring 和 Hibernate 是比较优秀的开源框架。

10.6 作　　业

一、选择题

1. 以下 EL 表达式的语法结构正确的是(　　)。

A. $[user. userName]　　B. #[user.userName]

C. $ {user. userName}　　D. # {user. userName}

2. 关于“.”操作符和“[ ]”操作符，以下说法不正确的是(　　)。

A. ${user.name}等价于 ${user[name]}

B. ${ user.name}等价于 ${user['name']}

C. 如果 user 是一个 List，则 ${ user[0]}的写法是正确的

D. 如果 user 是一个数组，则 ${ user[0]}的写法是正确的

3. 如果想在 JSP 页面声明一个名字为 name 的变量，应该使用(　　)标签。

A. <c:if>　　B. <c:set>　　C. <c:out>　　D. <c: forEach>

4. 如果要遍历一个数组中的所有元素，需要(　　)标签。

A. <c:if>　　B. <c:set>　　C. <c: remove>　　D. <c: forEach>

二、简答题

1. JSTL 常用的标签有哪些?

2. 请列举三个 Java Web 开发中比较优秀的开源框架?

3. EL 表达式的作用域访问对象有几个? 如何使用?

三、程序题

编写 JSP 页面使用 JSTL 和 EL 显示学生姓名列表，数据库为第 4 章课后作业的程序题的 school。